

S O F T W A R E

NEU!
Version 2.0

AMIGA

HiSoft

Devpac Assembler *deutsch*

Entwicklungspaket mit integriertem Editor/Assembler,
symbolischem Debugger und schnellem Linker
zum Einbinden von Hochsprachen-Modulen.
Erzeugt direkt ausführbare Programme.

3½"-Diskette



Markt & Technik

HiSoft

AMIGA

HiSoft Devpac 2.0 Assembler *deutsch*

Entwicklungspaket mit integriertem
Editor/Assembler, symbolischem Debugger
und schnellem Linker zum Einbinden von
Hochsprachen-Modulen. Erzeugt direkt
ausführbare Programme.

3 1/2"-Diskette



HiSoft

Markt&Technik Verlag Aktiengesellschaft · Hans-Pinsel-Straße 2 · 8013 Haar bei München

Die Informationen in diesem Produkt werden ohne Rücksicht auf einen eventuellen Patentschutz veröffentlicht.

Warennamen werden ohne Gewährleistung der freien Verwendbarkeit benutzt.

Bei der Zusammenstellung von Texten und Abbildungen wurde mit größter Sorgfalt vorgegangen.

Trotzdem können Fehler nicht vollständig ausgeschlossen werden.

Verlag, Herausgeber und Autoren können für fehlerhafte Angaben und deren Folgen weder eine juristische Verantwortung noch irgendeine Haftung übernehmen.

Für Verbesserungsvorschläge und Hinweise auf Fehler sind Verlag und Herausgeber dankbar.

Alle Rechte vorbehalten, auch die der fotomechanischen Wiedergabe und der Speicherung in elektronischen Medien.

Die gewerbliche Nutzung der in diesem Produkt gezeigten Modelle und Arbeiten ist nicht zulässig.

Amiga, AmigaDOS, Amiga Kickstart, Amiga Workbench und Amiga Assembler sind eingetragene Warenzeichen von Commodore-Amiga Inc., USA

Devpac Amiga, GenAm und MonAm sind eingetragene Warenzeichen von HiSoft

Motorola ist ein eingetragenes Warenzeichen von Motorola Inc.

15 14 13 12 11 10 9 8 7 6 5 4 3

92 91 90 89

Bestell-Nr. 54131

© 1989 by HiSoft, Great Britain

© 1989 der deutschen Übersetzung bei Markt & Technik Verlag Aktiengesellschaft,
Hans-Pinsel-Straße 2, D-8013 Haar bei München/West-Germany

Alle Rechte vorbehalten

Einbandgestaltung: Grafikdesign Heinz Rauner

Druck: Huber, Dießen

Printed in Germany

Inhaltsverzeichnis

Kapitel 1 Einführung

Vorwort des Übersetzers	9
1.1 Was ist Devpac Amiga?	11
1.2 Copyright	11
1.3 Anfertigen einer Sicherungskopie	11
1.4 Die Readme-Datei	12
1.5 Einführung in das CLI	12
1.6 Details zum Anfertigen einer Sicherungskopie	13
1.6.1 Amiga mit einem Diskettenlaufwerk	13
1.6.2 Amiga mit zwei Diskettenlaufwerken	13
1.7 Einrichten einer Arbeitsdiskette	13
1.7.1 Systeme mit einem Laufwerk	14
1.7.2 Systeme mit zwei Laufwerken	14
1.7.3 Für Systeme mit Festplatte	15
1.8 Der Entwicklungs-Zyklus	16
1.9 Inhalt der Diskette	16
1.9.1 Diskette 1	17
1.9.2 Diskette 2	17
1.10 Zum Gebrauch dieses Handbuchs	18
1.10.1 Benutzer von Devpac Amiga, Version 1	18
1.10.2 Anfänger	18
1.10.3 Erfahrene Benutzer	18
1.11 Eine Kurzeinführung	19

Kapitel 2	2.1	Einführung	23
Der Editor/	2.2	Der Bildschirm-Editor	24
Assembler	2.2.1	Einführung	24
	2.2.2	Einige Worte über Requester	25
	2.2.3	Datei-Requester	26
	2.3	Eingabe von Text und Cursor-Bewegungen	26
	2.3.1	Cursortasten	27
	2.3.2	Tabulator-Taste	28
	2.3.3	Backspace-Taste	28
	2.3.4	Delete-Taste	28
	2.3.5	Zeile ...	28
	2.3.6	Dateianfang	28
	2.3.7	Dateiende	29
	2.4	Verlassen von GenAm	29
	2.5	Löschen von Text	29
	2.5.1	Löschen einer Zeile	29
	2.5.2	Löschen bis zum Ende einer Zeile	29
	2.5.3	Gelöschte Zeile wieder einfügen	29
	2.5.4	Gesamten Text löschen	29
	2.6	Disketten-Operationen	30
	2.6.1	Text sichern	30
	2.6.2	Sichern	30
	2.6.3	Text laden	30
	2.6.4	Text-File einfügen	31
	2.6.5	Verzeichnis ändern	31
	2.7	Suchen und Ersetzen	31
	2.8	Block-Kommandos	32
	2.8.1	Markieren eines Blocks	32
	2.8.2	Sichern eines Blocks	32
	2.8.3	Kopieren eines Blocks	32
	2.8.4	Löschen eines Blocks	33
	2.8.5	Block in den Blockspeicher kopieren	33
	2.8.6	Block einfügen	33
	2.8.7	Drucken eines Blocks	33
	2.9	Verschiedenes	34
	2.9.1	Hilfsbildschirm	34
	2.9.2	Voreinstellungen	34
	2.10	Assemblieren und Programm ablaufen lassen	35
	2.10.1	Assemblieren	35

	2.10.2	Ablauf von Programmen	36
	2.10.3	Debuggen	36
	2.10.4	MonAm	36
	2.10.5	Anspringen der Fehlerzeile	37
	2.11	Das Editor-Fenster	37
Kapitel 3			
Der Makro-			
Assembler			
	3.1	Einführung	39
	3.2	Den Assembler aufrufen	39
	3.2.1	Vom Editor aus	39
	3.2.2	Der Stand-alone-Assembler	41
	3.3	Binäre Dateitypen	43
	3.4	Codearten	43
	3.5	Assembler-Befehlsformat	44
	3.5.1	Das Label-Feld (Markenfeld)	44
	3.5.2	Das Mnemonik-Feld	45
	3.5.3	Das Operanden-Feld	45
	3.5.4	Das Kommentar-Feld	45
	3.5.5	Ausdrücke	46
	3.5.6	Lokale Labels	49
	3.6	Der Befehlssatz	50
	3.6.1	Wortgrenzen	50
	3.7	Assembler-Direktiven	52
	3.7.1	Assembler-Kontrolle	52
	3.7.2	Zusammenfassung der Optionen	56
	3.7.3	Listing-Kontrolle	59
	3.7.4	Label-Direktiven	61
	3.7.5	Bedingtes Assemblieren	63
	3.8	Makro-Operationen	65
	3.9	Ausgabedatei-Formate	70
	3.9.1	Das richtige Format	70
	3.10	Ausgabedatei-Direktiven	71
	3.10.1	Sektionen	71
	3.10.2	Importe und Exporte	72
	3.11	Zusammenfassung der Direktiven	74
	3.12	Der Linker BLink	76
	3.12.1	Maschinen mit einem Laufwerk	76
	3.12.2	Maschinen mit zwei Laufwerken	76

Kapitel 4	4.1	Einführung	77
MonAmiga,	4.2	Exceptions des 68000	77
der symbolische	4.3	Starten von MonAm	79
Debugger	4.4	Programme debuggen	79
	4.5	MonAm und Multitasking	79
	4.6	Speicher ansehen	80
	4.7	Symbolisch debuggen	80
	4.8	Dialog- und Alarmboxen	80
	4.9	Das Anfangs-Display	81
	4.10	Das Haupt-Display	81
	4.10.1	Benutzung der Fenster	81
	4.11	Eingabe von Befehlen	82
	4.12	Überblick über MonAm	82
	4.13	MonAm – Referenzteil	84
	4.13.1	Numerische Ausdrücke	84
	4.13.2	Fenstertypen	85
	4.13.3	Fenster-Befehle	87
	4.13.4	Bildschirmumschaltung	89
	4.13.5	Breakpoints	90
	4.13.6	History	92
	4.13.7	MonAm verlassen	92
	4.13.8	Laden und Abspeichern	93
	4.13.9	Programme ausführen	94
	4.13.10	Speicher durchsuchen	95
	4.13.11	Verschiedenes	96
	4.13.12	Zusammenfassung der MonAm-Kommandos	99
	4.14	Fehlersuche	100
	4.14.1	Beschränkungen	100
	4.14.2	Die Bug-Jagd	101
	4.14.3	Exception-Analyse	102

Anhang A	AmigaDOS-Fehlernummern	103
-----------------	------------------------	-----

Anhang B	B.1	Fehlermeldungen	105
GenAm	B.2	Warnungen	112

**Anhang C
Aufruf von
System-Routinen**

C.1	Einführung	113
C.2	Libraries	113
C.2.1	Diskfont-Library	115
C.2.2	DOS-Library	115
C.2.3	Exec-Library	115
C.2.4	Graphics-Library	116
C.2.5	Icon-Library	116
C.2.6	Intuition-Library	116
C.2.7	Maths-Libraries	116
C.3	Beispielprogramme	117
C.3.1	demo.s	117
C.3.2	freemem.s	117
C.3.3	helloworld.s	117
C.4	Vergleich CLI/Workbench	117
C.4.1	CLI-Start	118
C.4.2	Workbench-Start	118
C.5	Andere 68000-Prozessoren	118

**Anhang D
Gebrauch
des CLI**

D.1	Einführung	121
D.2	Files, Laufwerke und Directories	121
D.3	AmigaDOS-Wildcards	123
D.4	Gerätenamen	123
D.4.1	RAM:	123
D.4.2	RAD:	124
D.4.3	PAR:	124
D.4.4	SER:	124
D.4.5	PRT:	124
D.4.6	NIL:	124
D.4.7	CON:	124
D.4.8	RAW:	125
D.4.9	*	125
D.5	AmigaDOS-Befehle	125
D.6	Startup-Sequence	130

**Anhang E
Konvertierung
von anderen
Assemblern**

E.1	Amiga-Makro-Assembler und MCC-Assembler	131
E.1.1	K-Seka	132
E.2	Neue Commodore-Include-Files	132
E.2.1	ConvertI	132
E.2.2	ConvertFD	133

**Anhang F
Verbesserungen
in der
Version 2.0**

F.1	Der Editor	135
F.1.1	Der Assembler	135
F.1.2	Der Debugger	137
F.1.3	Integration	137

	Stichwortverzeichnis	139
--	----------------------	-----

	Hinweise auf weitere Markt&Technik-Produkte	142
--	---	-----

Vorwort des Übersetzers

Bevor ich das Amiga-Assembler-Buch schrieb, habe ich acht verschiedene Assembler getestet und davon drei, nämlich Metacomco, SEKA und Devpac, in die engere Wahl gezogen.

Nach einem ausführlichen Test dieser drei stand für mich als eindeutiger Sieger Devpac von HiSoft fest. Ich habe daraufhin die zahlreichen Listings des Buches mit Devpac entwickelt und immer gehofft, daß auch die Leser meiner Meinung sein werden.

Deshalb freut es mich besonders, daß sich ein so führender Software-Spezialist wie Markt&Technik dafür entschieden hat, dieses Produkt in sein Angebot aufzunehmen, womit meine Einschätzung dieses Assemblers bestätigt wurde.

Das war für mich auch der Grund, die Übersetzung des Handbuches zu übernehmen, obwohl ich sonst viel lieber kreativ als Programmierer und Autor arbeite.

Ganz konnte ich diese »Kreativität« auch bei der Übersetzung nicht unterdrücken, weshalb dies keine Wort-für-Wort-Übersetzung geworden ist. Ich habe einige Inkonsistenzen beseitigt und einige Passagen, die meiner Meinung nach Einsteiger nicht auf Anhieb verstehen, etwas klarer ausgedrückt, als es das Original tut. Andererseits bin ich schon der Ansicht, daß im Original einige Passagen sehr ausführlich beschrieben sind, hier habe ich dennoch nichts gekürzt und nichts weggelassen.

Einige Wörter habe ich nicht übersetzt. Dabei handelt es sich um Fachausdrücke aus der Assembler-Programmierung und um Begriffe aus der Amiga-Welt. Die Assembler-Sprache selbst ist nun einmal englischer Klartext, da würde ein Eindeutschen nur die Verbindung zur Praxis aufheben. Sinngemäßes gilt für die Amiga-Welt. Die vielen hundert Funktionen, die Ihnen das Betriebssystem zur Verfügung stellt, haben englische Namen. Das gilt auch für alle CLI-Kommandos. Hier würde zum Beispiel das Wort »Inhaltsverzeichnis« anstatt »Directory« die Verbindung zu Kommandos wie »dir« oder »mkdir« aufheben.

Eines fehlt im Buch ganz, nämlich die Erläuterung sehr häufig gebrauchter Namen. »Devpac« ist das Kürzel für »Development Package« also Entwicklungspaket. Tatsächlich erhalten Sie mit Devpac ein Paket bestehend aus Editor, Assembler und Debugger (und einigen Zugaben).

MonAm oder MonAmiga ist das Kürzel für Monitor Amiga. Hierbei handelt es sich um einen symbolischen Debugger. Ich nenne MonAm auch Mon Ami (mein Freund), weil er mir schon aus so mancher Klemme geholfen hat.

Bliebe noch GenAm oder GenAmiga. Gen heißt Generator, genau Programm-Generator, womit die Kombination von Editor und Makro-Assembler gemeint ist. Da diese beiden wichtigsten Werkzeuge als integrierte Einheit ständig im RAM stehen, reicht ein Tastendruck im Editor, um den Assembler aufzurufen. Dieser merkt sich bis zu 30 Fehlerstellen, womit man dann im Editor von Fehlerzeile zu Fehlerzeile gehen und verbessern kann. Was dann jeweils falsch ist, steht sogar in der Statuszeile.

Dieser Bedienkomfort verbunden mit hohem Tempo und allen Leistungsmerkmalen guter Assembler hat mich für Devpac begeistert und tut es immer noch.

Ich wünsche Ihnen viel Erfolg bei der Entwicklung Ihrer Programme. Die besten Voraussetzungen dafür haben Sie mit Ihrer Entscheidung für Devpac bereits geschaffen.

Happy Programming

Ihr

Peter Wollschlaeger

Kapitel

1

Einführung

1.1 Was ist Devpac Amiga?

Dexpac Amiga ist ein superschnelles Assembler-Entwicklungspaket für alle Commodore-Computer. Es enthält alle Programme, die Sie zur Entwicklung von Programmen in Maschinensprache benötigen. Darin enthalten sind ein Makro-Assembler, ein Bildschirm-Editor, ein Linker und Include-Dateien, mit denen Sie eine Verbindung mit dem Betriebssystem des Amiga herstellen können.

1.2 Copyright

Die Software von Dexpac wird auf einer nicht kopiergeschützten Diskette ausgeliefert. Software und Handbuch unterliegen dem deutschen Urheberrecht. Jegliche Vervielfältigung, auch auszugsweise, ist nur nach schriftlicher Genehmigung des Verlages gestattet. Hiervon ausgenommen ist lediglich das Anfertigen von Sicherungsdisketten ausschließlich für den eigenen Bedarf. Die Weitergabe an Dritte ist nicht zulässig.

1.3 Anfertigen einer Sicherungskopie

Bevor Sie Dexpac einsetzen, sollten Sie von der mitgelieferten Diskette eine Kopie erstellen und das Original an einem sicheren Platz verwahren. Es ist nicht kopiergeschützt, um Ihnen ein einfaches Arbeiten zu ermöglichen. Bevor Sie mit dem Kopieren beginnen, aktivieren Sie den Schreibschutz Ihrer Originaldiskette, um ihre versehentliche Zerstörung zu vermeiden. Die genaue Kopiermethode hängt davon ab, wie viele Laufwerke Sie besitzen. Aber alle Methoden machen es erforderlich, daß Sie im CLI sind und mit dem ins interne Laufwerk eingelegten schreibgeschützten Original beginnen.

1.4 Die Readme-Datei

Wie alle HiSoft-Produkte wird auch Devpac laufend verbessert, und die neuesten Informationen, die noch nicht in das Handbuch aufgenommen werden konnten, werden dann in der Readme-Datei bekanntgegeben. Wenn Sie noch nicht im CLI sind, sollten Sie warten, bis alle Diskettenaktivitäten aufhören. Nehmen Sie dann alle Disketten heraus, und drücken Sie die Control- und beide Amiga-Tasten gleichzeitig. Legen Sie jetzt die Devpac-Amiga-Diskette ein. Sobald Sie sich im CLI befinden, geben Sie

```
type readme 
```

ein. Das (durchlaufende) Display kann durch Drücken der rechten Maustaste angehalten werden. Nachdem die Taste losgelassen wurde, läuft der Text weiter ab.

1.5 Einführung in das CLI

Wie die meisten Entwicklungstools ist auch Devpac für den Betrieb im CLI (Command Line Interface) ausgelegt. Wenn Sie die Diskette von der Workbench aus inspizieren, werden Sie deshalb nicht sehr viele Icons sehen (genauer gesagt, keine von Devpac-Files). Das CLI ist für den Normalanwender versteckt, aber als Programmierer brauchen Sie es. Die mitgelieferte Diskette 1 ist eine modifizierte Workbench-Diskette. Wenn Sie nach dem Einschalten (beim Amiga 1000 nach dem Durchlauf der Kickstart-Diskette) nach der Workbench-Diskette gefragt werden, legen Sie die Devpac-Diskette ein.

Wenn Ihr Amiga schon mit einer anderen Workbench-Diskette gestartet wurde, warten Sie eventuelle Disk-Aktivitäten ab, legen dann die Devpac-Diskette ein und machen einen Reset, was durch gleichzeitiges Drücken der -Taste und der beiden Amiga-Tasten erreicht wird. Das CLI entspricht der Benutzeroberfläche anderer nicht grafisch orientierter Computer, wie zum Beispiel MS-DOS oder CP/M. Es verlangt die Eingabe von Kommandos wie

```
dir 
```

um eine Directory-Anzeige zu erhalten. Eine Beschreibung darüber, wie das CLI benutzt wird, finden Sie in Anhang D.

1.6 Details zum Anfertigen einer Sicherungskopie

Bevor Sie Sicherungskopien erstellen, sollten Sie sich vergewissern, daß Ihre Master-Disketten schreibgeschützt sind.

1.6.1 Amiga mit einem Diskettenlaufwerk

Geben Sie nach dem Booten mit der Devpac-Amiga-Diskette

diskcopy df0: to df0:

ein. Dann folgen Sie den Anweisungen, die auf dem Bildschirm erscheinen. Die »disk to copy from« ist unsere Master-Diskette, die »disk to copy to« sollte eine neue, leere Diskette sein. Wenn die Kopie fertig ist, legen Sie die Devpac-Amiga-Disk 2 ein und geben

diskcopy df0: to df0:

ein. Befolgen Sie dann die Anweisungen auf dem Bildschirm. Die zu kopierende Diskette ist die Master-Diskette 2, die Diskette, auf die kopiert wird, sollte eine leere Diskette sein.

1.6.2 Amiga mit zwei Diskettenlaufwerken

Geben Sie

diskcopy df0: to df1:

ein. Legen Sie dazu die neue, leere Diskette in das externe Laufwerk. Drücken Sie dann , so daß eine Kopie von Diskette 1 gemacht wird. Wenn die Kopie angefertigt ist, legen Sie die Master-Disk 2 und eine leere Diskette in das externe Laufwerk ein. Folgen Sie dann den Anweisungen auf dem Bildschirm. Nun haben Sie eine Sicherungskopie. Warten Sie ab, bis alle Diskettenaktivitäten beendet sind, nehmen Sie die Diskette(n) heraus, und verwahren Sie das Original an einem sicheren Ort. Legen Sie nun die neue Kopie ein, und erzeugen Sie einen Reset. () und beide Amiga-Tasten gleichzeitig drücken.)

1.7 Einrichten einer Arbeitsdiskette

Obwohl wir eine Sicherungskopie haben (die haben Sie doch erstellt, oder?), hat diese nicht genügend Kapazität zum effektiven Programmieren. Deshalb sollte eine Arbeitsdiskette angelegt werden. Die genaue Vorgehensweise hängt von Ihrer Hardware-Konfiguration ab.

1.7.1 Systeme mit einem Laufwerk

Bei Systemen mit nur einem Laufwerk ist es das beste, möglichst viel von der Workbench-Disk zu löschen. Dafür kommen zunächst alle .INFO-Files (welche für die Icon-Darstellung gebraucht werden), Fonts und Druckertreiber in Frage. Die folgenden Kommandos sind (auf einer weiteren Sicherungskopie) dafür erforderlich:

```
cd df0: 
delete #?.info 
delete devs/printers 
delete dir 
```

Jetzt wird eine Liste von Druckertreibern ausgegeben. Löschen Sie mit dem Delete-Kommando diejenigen, die Sie nicht brauchen. Wenn trotzdem noch zu wenig Platz vorhanden ist, löschen Sie am besten das EXAMPLES-Verzeichnis, selten gebrauchte Include-Dateien und einige Kommandos des C:-Verzeichnisses.

1.7.2 Systeme mit zwei Laufwerken

Besitzer von Systemen mit zwei Laufwerken sollten die Workbench freihalten von Source-Code und diese Dateien mit den Include-Dateien auf Diskette im externen Laufwerk speichern. Die Kommandos dafür sind:

```
format drive df1: name "Source" 
```

Wichtig: Der erste Befehl formatiert eine neue Diskette im externen Laufwerk und vernichtet damit alle darauf befindlichen Daten. Wenn Sie eine vorformatierte Diskette benutzen, brauchen Sie diesen Befehl nicht auszuführen.

Legen Sie eine neue Diskette in das externe Laufwerk, und geben Sie dann ein, um sie zu formatieren. Nun folgt:

```
cd df0: 
mkdir df1:include 
mkdir df1:examples 
copy include to df1:include all 
copy examples to df1:examples all 
delete include all 
delete examples all 
```

Die Kommandos schaffen Directories auf dem externen Drive für die Beispielpprogramme (in EXAMPLES) und die Include-Files. Dann löschen Sie diese von Ihrer Arbeitskopie im internen Laufwerk.

Sie wollen nun bestimmt die Programm-Dateien von der Devpac-Amiga-Diskette 2 auf Ihre Workbench kopieren. Legen Sie deshalb die Sicherungskopie der Devpac-Amiga-Master-Diskette 2 in df1: und geben Sie

```
copy df1:c to df0:c all 
```

ein. Nach

```
cd df1: 
```

wird auf das externe Laufwerk umgeschaltet, so daß Sie Ihre Quellfiles einfacher (ohne Angabe des Pfadnamens) editieren können. Es ist oft nützlich, ein solches Kommando in das Startup-File einzubauen.

1.7.3 Für Systeme mit Festplatte

Festplattenbesitzer sollten zuerst von der Harddisk booten und nicht von der Devpac-Diskette. Danach sollte die Devpac-Diskette 1 in das interne Laufwerk eingelegt und die folgenden Befehle ausgeführt werden, um die Programme auf die Harddisk zu kopieren:

```
copy df0:c/genam2 to c: 
```

```
copy df0:c/monam2 to c: 
```

```
copy df0:c/genim2 to c: 
```

Als nächstes sollten die beiden Subdirectories kopiert werden, doch das hängt von Ihren Anforderungen ab. Sie könnten sich zum Beispiel ein gesondertes Devpac-Directory anlegen. Was immer Sie auch wählen – Sie müssen die Include- und Example-Directories komplett kopieren. Wenn Sie sich für ein gesondertes Devpac-Directory entschieden haben, sind folgende Kommandos erforderlich (angenommen, Ihre Harddisk heißt dh0:)

```
makedir dh0:devpac 
```

```
copy df0:include to dh0:devpac all 
```

```
copy df0:examples to dh0:devpac all 
```

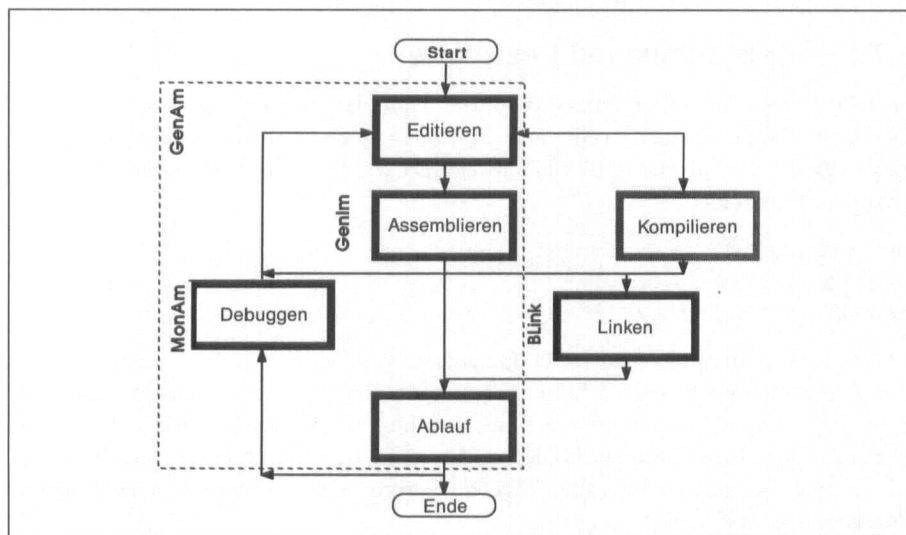
Beachten Sie, daß Sie die INCDIR-Direktive entsprechend ändern müssen, wenn Sie das Include-Directory in ein anderes Directory gepackt haben. Sie können auch die folgende Datei von Ihrer Devpac-Amiga-Diskette 2 kopieren:

```
copy df0:c/blink to dh0:c
```

Andere Dateien brauchen Sie erst dann kopieren, wenn sie benötigt werden.

1.8 Der Entwicklungs-Zyklus

Das Devpac-Entwicklungspaket dient zur Eingabe des Assembler-Quelltextes, zur Assemblierung in Maschinen-Code und zum Debuggen, wenn ein Programm nicht (oder nicht so wie geplant) laufen sollte. Abhängig von der jeweiligen Anwendung können Sie auch einen Linker benutzen, um zum Beispiel mehrere Module zusammenzubinden. Auch Module, die von Compilern höherer Programmiersprachen generiert wurden, können so mit Assembler-Modulen gebunden werden. Der normale Entwicklungszyklus kann am besten mit einem Diagramm wie dem folgenden dargestellt werden. Devpac wurde entwickelt, um diesen Ablauf so schnell und effektiv wie möglich bewältigen zu können.



Die Linkfunktion ist optional, der Assembler kann auch sofort ausführbaren Code generieren. Natürlich wird ein Compiler (wie im Bild angedeutet) nicht mitgeliefert, es können aber alle gängigen verwendet werden.

1.9 Inhalt der Diskette

Die erste der zwei beigelegten 3,5"-Disketten ist eine Standard-Workbench-Diskette, die so konfiguriert ist, daß der Amiga nach dem Einschalten (oder einem Reset) im CLI ist. Die zweite Diskette enthält zusätzliche Dateien.

1.9.1 Diskette 1

Programme (im Unterverzeichnis C/):

GenAm2	der Bildschirmeditor und das Kontrollprogramm
MonAm2	der Disassembler/Debugger
GenIm2	der Assembler

Sonstige Dateien

libs/libfile.monam	vom Debugger verwendete Binärdatei
libs/arp.library	Zusatz-Library
Readme	letzte Informationen zu Devpac Amiga

Zusätzliche Unterverzeichnisse

include	beinhaltet Dateien für den Zugriff auf das Betriebssystem, Beschreibung in Anhang C.2
examples	der Source-Code zu verschiedenen Beispielprogrammen, Beschreibung in Anhang C.3

1.9.2 Diskette 2

Programme (im Unterverzeichnis C/)

BLink	Public-domain-Linker (siehe Kapitel 3.12)
ConvertFD	Utility zum Konvertieren von FD-Dateien
ConvertI	Utility zum Konvertieren von Include-Dateien
einige Workbench-Dateien,	die auf Disk 1 nicht genügend Platz haben

Textdateien

blink.doc	Anweisungen zum Gebrauch des Linkers
-----------	--------------------------------------

Zusätzliche Unterverzeichnisse

include.cbm	die Original-Commodore-Include-Dateien mit Kommentierung
include.strip	die Original-Commodore-Include-Dateien ohne Kommentierung
fd.files	die Original-Commodore-fd-Dateien beinhaltet, welche Register beim Aufruf von Libraries gebraucht werden
library	beinhaltet amiga.lib
arp	beinhaltet Teile von Charlie Heaths AmigaDOS-Replacement-Projekt (ARP)

1.10 Zum Gebrauch dieses Handbuchs

Dieses Handbuch macht keinen Versuch, die 68000-Programmierung zu lehren oder auf Einzelheiten des 68000-Befehlssatzes einzugehen. Auf geeignete Literatur wird im Anhang verwiesen. Die Anhänge geben einen Überblick über die technischen Zusammenhänge des Amiga, sie sind aber nicht als komplette technische Beschreibung der Maschine gedacht. Dieses Handbuch besteht aus vier Teilen: der Einführung, einem Kapitel über den Editor/Assembler, einem Kapitel über den Makro-Assembler und einem Kapitel über den Debugger. Zusätzlich gibt es noch sechs Anhänge, die verschiedene Informationen beinhalten. Ihr Gebrauch des Handbuchs hängt davon ab, welcher Benutzertyp Sie sind.

1.10.1 Benutzer von Devpac Amiga, Version 1

Gehen Sie in Anhang F und lesen Sie das Kapitel, in dem die neuen Features beschrieben werden. Wenn Sie MonAm benutzen wollen, sollten Sie Kapitel 4 durcharbeiten, denn MonAm wurde beträchtlichen Veränderungen unterworfen. Danach werden Sie vermutlich das Handbuch auf Neuerungen durchsehen.

1.10.2 Anfänger

Am Ende dieses Kapitels finden Sie eine einfache Einführung, die Sie nachvollziehen sollten, um mit den Hauptteilen des Programms vertraut zu werden. In Kapitel 2 wird der Editor behandelt, in Kapitel 3 wird der Assembler vorgestellt. Dessen Funktionen werden Ihnen klarer, wenn Sie einmal ein erfahrener Devpac-Anwender sind. Besonders ans Herz legen wollen wir Ihnen die Übersicht über den Debugger in Kapitel 4. Ein Blick auf den mitgelieferten Source-Code könnte hilfreich sein.

1.10.3 Erfahrene Benutzer

Wenn Sie bereits Erfahrungen in der 68000-Assemblierung sammeln konnten, aber noch nicht mit Devpac Amiga in Berührung gekommen sind, geben wir Ihnen hier einen kurzen Überblick, wie eine Quellcode-Datei assembliert wird:

Laden Sie GENAM2.PRG, drücken Sie **A-L** und suchen Sie die Datei aus, die Sie in den Editor laden wollen. Drücken Sie **A-A** und wählen Sie die entsprechenden Optionen aus – wenn Sie ausführbaren Code erzeugen wollen, klicken Sie auf den **Speicher**knopf, um eine höhere Geschwindigkeit zu erzielen. **Return** startet den Assembler, das Drücken einer beliebigen Taste hält ihn an, durch **Return** kehren Sie in den Assembliervorgang zurück. Assemblierfehler werden gespeichert. Wenn Sie in den Editor zurückkehren, steht der Cursor auf dem ersten Fehler. Durch Eingabe

von **A**-**J** finden Sie auch weitere Fehler. Um das erfolgreich assemblierte Programm ablaufen zu lassen (vorausgesetzt, Sie haben in den Speicher assembliert), geben Sie **A**-**X** ein. Wenn Sie auf Diskette assembliert haben, verlassen Sie den Editor und lassen es vom CLI aus ablaufen. Als Schnelleinführung zum Debugger empfehlen wir Ihnen das folgende Kapitel.

1.11 Eine Kurzeinführung

Das folgende Kapitel ist eine kurze Einführung mit dem Ziel, Ihnen zu zeigen, wie einfach und schnell das Editieren, Assemblieren und Debuggen mit Devpac ist. In diesem Tutorial werden wir ein einfaches Programm assemblieren und debuggen. Das Programm selbst soll nur eine kurze Meldung auf dem Bildschirm ausgeben. Um diesem Tutorial folgen zu können, müssen Sie bereits im CLI sein. Wenn dies nicht der Fall sein sollte, so warten Sie alle Diskettenaktivitäten ab und nehmen alle Disketten heraus. Drücken Sie die **Ctrl** und beide Amiga-Tasten gleichzeitig, und legen Sie dann Ihre laut Abschnitt 1.6 erstellte Kopie ein. Um die Arbeit zu erleichtern, wird das Directory, in dem sich der Quelltext befindet, zum aktuellen Directory gemacht. Tippen Sie dazu

```
cd examples Return
```

ein. Als nächstes laden Sie den Editor/Assembler durch die Eingabe von

```
genam2 Return
```

Nach kurzer Zeit erscheint ein leeres Fenster. Um nun ein File zu laden, drücken Sie die rechte Maustaste, bewegen dann den Mauszeiger auf das *Datei*-Menü und wählen dort *Laden ...* aus. Es erscheint ein File-Requester. Klicken Sie das Namensfeld an und geben Sie den zu ladenden File-Namen ein, hier

```
demo.s Return
```

In dieser Kurzeinführung werden wir ein einfaches Programm mit zwei Fehlern assemblieren und debuggen. Das Programm selbst soll lediglich eine Meldung ausgeben. Wenn das File geladen ist, zeigt das Fenster die ersten Programmzeilen. Wenn Sie einen schnellen Blick auf das ganze Programm werfen wollen, können Sie die Tastenkombination **Shift** und **↓** oder **Ctrl**-**C** drücken, um eine Seite weiterzublätern.

Bei den meisten kurzen Programmen ist es am besten, wenn man sie versuchsweise assembliert, ohne gleich List-Files und Object-Files zu erzeugen. Damit ist eine schnelle Kontrolle auf Syntaxfehler, Tippfehler und ähnliches möglich.

Bewegen Sie den Mauszeiger auf das *Programm*-Menü, und wählen Sie *Assemblieren* aus. Eine Dialogbox erscheint, in der nichts verändert werden braucht, außer daß der Knopf *Nein* angeklickt werden sollte. Klicken Sie dann auf *Assemblieren*, oder geben Sie Return ein, worauf der Assembliervorgang beginnt.

Der Assembler zeigt den Fehler »Unbekannte Anweisung« an. Durch Drücken einer beliebigen Taste kehren Sie in den Editor zurück. Der Cursor steht darauf in der fehlerhaften Zeile, die Fehlermeldung wird in der Statuszeile angezeigt. Die Programmzeile »MOV.L« sollte in »MOVE.L« geändert werden. Nachdem Sie das getan haben, klicken Sie wieder auf *Assemblieren* im *Programm*-Menü. Wählen Sie diesmal *Speicher*, das Programm wird dann anstatt auf Diskette in den Speicher assembliert. Dadurch wird die Geschwindigkeit des Assemblers erhöht, und Probelaufe können sofort ausgeführt werden. Erneutes Klicken auf *Assemblieren* läßt den Assemblier-Vorgang weiterlaufen. Nach dem Drücken einer Taste kehren Sie in den Editor zurück und könnten das Programm nun ablaufen lassen. Zwar wurde diesmal ordnungsgemäß assembliert, jedoch ist das Programm noch nicht lauffähig. Sie würden vermutlich die Meldung »Software error task held« vom System erhalten. Darauf müßten Sie neu booten. Um Programme zu prüfen und Fehler zu finden, brauchen Sie den Debugger. Klicken Sie also auf *Debuggen* im *Programm*-Menü. Jetzt wollen wir den Debugger lediglich dazu einsetzen, um herauszufinden, ob Fehler vorhanden sind, und um deren Ursache zu ergründen. Drücken Sie also Ctrl-R. Nach einer kurzen Verzögerung erscheint am unteren Rand des Bildschirms die Meldung »Adreßfehler«, das Disassemblierungsfenster zeigt die Anweisung

```
start MOVE.L dosname,a1
```

Dadurch wird ein Adreßfehler erzeugt, denn »dosname« befindet sich in einer Adresse, auf die mit der »MOVE.L«-Anweisung nicht zugegriffen werden kann. Vor »dosname« sollte nämlich »#« stehen, damit die Adresse von »dosname« ins Register »a1« geschrieben werden kann. Geben Sie zuerst Ctrl-Q ein, um in den Editor zurückzukehren, und danach Ctrl-C, um das Debuggen zu beenden. Den Fehler können wir im Source-Code beseitigen. Geben Sie A-T ein, um an den Dateianfang zu gelangen, und klicken Sie auf *Suchen* im *Suchen*-Menü. Um die fehlerhafte Anweisung zu finden, geben Sie

```
move.l
```

ein und drücken dann Return, um die Suche zu starten. Gleich das erste Vorkommen ist dasjenige, nach dem wir suchen:

```
start move.l dosname,a1
```

Wir ändern dieses jetzt in

```
start move.l # dosname, a1
```

und assemblieren wieder. Wenn Sie dann auf *Ausführen* im *Programm*-Menü klicken, sehen Sie die angekündigte Meldung. Durch Drücken einer beliebigen Taste kehren Sie in den Editor zurück. Obwohl das Programm jetzt läuft, benutzen wir MonAm, den Debugger, um das Programm Schritt für Schritt zu durchlaufen. Klicken Sie dazu auf *Debuggen* im *Programm*-Menü, daraufhin erscheint der Debugger mit der Meldung »Breakpoint« und zeigt Ihr Programm. Das obere Fenster zeigt die Maschinenregister an, das zweite die Disassemblierung des Programms und das dritte weitere Speicher. In Fenster 2, dem Disassemblierungsfenster, sehen Sie die aktuelle Anweisung, in unserem Fall

```
start move.l # dosname, a1
```

Da im Source-Code die Debug-Option spezifiziert wurde, erscheinen die Symbole im Debugger. Sehen wir uns nun die Umgebung von »dosname« an. Geben Sie »A3« an, erscheint die Überschrift von Fenster 3 invertiert. Danach tippen Sie **[A]**. In die erscheinende Dialogbox mit der Frage »Window start address?« geben Sie

```
string
```

(in Kleinbuchstaben) ein und **[Return]**. Dadurch erscheint an dieser Adresse Fenster 3, die Meldung wird in Hex und ASCII angezeigt. Um die Move-Anweisung auszuführen, geben Sie **[Ctrl]-[Z]** ein. Sie wird daraufhin ausgeführt, am Bildschirm erscheinen die neuen Werte des Programmzählers und des Registers A1. Nach abermaliger Eingabe von **[Ctrl]-[Z]** wird die MOVEQ-Anweisung ausgeführt und d0 wird modifiziert. Die beiden folgenden Anweisungen betreffen die Expansion des CALLEXEC-Open-Library-Makro-Aufrufs. Mit **[Ctrl]-[Z]** single-steppen Sie die »move to a6«-Anweisung. Als nächstes erscheint

```
JSR _LV00openLibrary(A6)
```

Das ist der Aufruf der Exec-Library. Alle Aufrufe des Amiga-Betriebssystems haben die gleiche Form. Hier brauchen wir nicht single-steppen, da wir wissen, daß Amiga-DOS funktioniert (zumindest meistens!). Um den Systemaufruf als eine Anweisung zu behandeln, benutzen Sie **[Ctrl]-[T]**. Single-steppen Sie nun TST.L D0. D0 sollte ungleich 0 sein; das Z-Flag ist deshalb nicht in der Registeranzeige von MonAm enthalten. Fahren Sie nun fort, **[Ctrl]-[Z]** einzugeben, bis Sie zu

```
JSR _LV00output(a6)
```

gelangen. Durch diesen Aufruf wird in der DOS-Library der Output-Handle gesucht. Mit **[Ctrl]-[T]** können Sie darüber hinweggehen. Single-steppen Sie weiter mit **[Ctrl]-[Z]**, bis Sie zu

`JSR _LV0Write(a6)`

kommen. Durch diesen Aufruf wird unser String auf den Bildschirm geschrieben – wir sollten uns davon überzeugen, daß die Register korrekt gesetzt wurden. d1 sollte den Output-Handle haben, der vom Output-Aufruf in d0 rückgemeldet wurde. d2 sollte die Adresse der Meldung beinhalten. In der Registeranzeige sehen Sie die ASCII-Bytes des Strings. Mit **[Ctrl]-[T]** übergehen Sie den Schreibaufruf. Um sicherzustellen, daß alles ordnungsgemäß arbeitet, geben Sie »V« ein. Der Programm-Bildschirm wird daraufhin angezeigt. Durch Eingabe einer beliebigen Taste kehren Sie in MonAm zurück. Das, was noch übrig ist, ist unsere Deinitialisierung. Mit **[Ctrl]-[Z]** und **[Ctrl]-[T]** können Sie sie durchlaufen. Die letzte Anweisung im Programm ist »RTS«. MonAm wird beendet, wenn RTS single-gestept wurde. Daraufhin kehrt man in den Editor zurück.

Kapitel

2

Der Editor/Assembler

2.1 Einführung

Für die Eingabe und das Assemblieren eines Programms brauchen Sie einen Editor und einen Assembler. GenAm ist der Editor, der den Assembler lädt und zum Laufen bringt, GenIm verleiht ihm das Erscheinungsbild eines Voll-Screen-Editors und eines Hochgeschwindigkeits-Assemblers in einem. Mit GenAm können Sie Ihre Programme auch direkt vom Speicher aus assemblieren, ohne das Programm verlassen zu müssen oder einen Diskettenzugriff zu machen und den Debugger per Tastendruck aufrufen zu müssen. Durch die Integration dieser Eigenschaften ist es möglich, Fehlerkorrekturen und Änderungen extrem schnell – da ja kein Diskettenzugriff erfolgt – zu vollziehen.

In diesem Kapitel wird erläutert, wie der Editor gebraucht wird und wie Programme assembliert werden. Der Assembler und Editor an sich werden in den nächsten Kapiteln behandelt. Um GenAm laufen zu lassen, geben Sie vom CLI aus GENAM2 ein. Wenn er geladen ist, wird ein Fenster geöffnet, in das Sie Einträge machen können. Sie können wahlweise eine Kommandozeile mit angeben, die den File-Namen eines zu editierenden Programms und verschiedene Optionen enthalten kann.

Option	Beispiel
-s Setze Arbeitsbereich für Editor	-s100 (ergibt 100 Kbyte)
-x Setze Maximum XREFs	-x250 (250 XREF erlaubt)
+c Groß-/Kleinbuchstaben-Unterscheidung	
-c Kleinbuchstaben	
+l Erzeugt linkbaren Code	
-l Erzeugt ausführbaren Code	

Die Bedeutung der Optionen wird Ihnen klarer, wenn Sie den Rest dieses Kapitels gelesen haben. Zum Beispiel startet das Kommando »genam2 examples/demo.s -s50 +c -l« GenAm, lädt das File demo.s in einen Arbeitspuffer von 50 Kbyte, beim Assemblieren wird Groß-/Kleinschrift unterschieden und ausführbarer Code erzeugt.

Die Beschreibung des Editors und des Assemblers erfolgt in zwei getrennten Abschnitten. Beginnen wir mit dem Editor.

2.2 Der Bildschirm-Editor

2.2.1 Einführung

Ein Text-Editor ist ein Programm, das Ihnen erlaubt, Textzeilen in den Speicher einzugeben, zu ändern, sie auf der Diskette zu speichern und von dort wieder zu laden. Es gibt zwei Arten von Editoren: zum einen Zeilen-Editoren, die jede Zeile einzeln behandeln und sehr schwierig zu bedienen sind, und zum anderen Bildschirm-Editoren, die Ihren Text auf einmal auf dem Schirm abbilden. Diese sind wesentlich einfacher zu bedienen.

Der Editor-Teil von GenAm ist ein Bildschirm-Editor, der Ihnen die Eingabe von Text, das Ändern, Speichern und Laden von der Diskette erlaubt. Er läßt Sie auch den ganzen Text oder Teile davon ausdrucken und kann Begriffe finden und durch andere ersetzen. Er basiert auf Intuition, was heißt, daß er die sehr bedienerfreundlichen Features des Amiga, die Ihnen von anderen Programmen her schon vertraut sind, bietet. Dazu gehören Fenster, Menüs und Mausbedienung. Wenn Sie jedoch die alten Methoden der Computer vor dem Fortschritt des WIMP gewohnt sind, so können Sie GenAm auch bedienen, ohne die Maus anfassen zu müssen.

Der Editor ist RAM-orientiert, was bedeutet, daß das ganze File immer im RAM steht und Sie nicht warten müssen, bis der Editor jeweils Textsegmente von der bzw. zur Disk umspeichert. Weil der Amiga so viel Speicherplatz besitzt, gibt es hier ein Speicherlimit, wie man es oft bei älteren Editoren findet, nicht. Wenn Sie genug RAM haben, können Sie Files von über 300 Kbyte editieren (stellen Sie aber sicher, daß dann auf der Diskette noch genug Platz zum Abspeichern ist). Weil auch alle Editor-Aktionen wie die Suchfunktion RAM-orientiert sind, laufen sie blitzschnell ab. Ein eingetipptes Programm bringt nicht viel, wenn Sie es nicht auf der Disk sichern können. Deshalb hat der Editor sehr vielfältige Optionen zum Laden und Speichern von Texten, was Ihnen erlaubt, Teile des Textes (oder den ganzen Text) zu sichern oder Textteile in die Mitte eines bestehenden Textes einzuladen, um nur einige Beispiele zu nennen.

Für die Bedienung des Editors stehen Ihnen verschiedene Methoden zur Verfügung, die Sie einzeln oder in Kombination anwenden können:

- Steuerung über Tasten, wie die Cursortaste oder eine Funktionstaste.
- Auswahl eines Menüpunktes wie zum Beispiel *Sichern* per Maus.
- Menükürzel, über die Kombination von rechter Amiga-Taste und einem Buchstaben.

Im folgenden wird die Kombination von rechter Amiga-Taste und Taste **X** mit **A-X** abgekürzt. Entsprechend wird die Kombination von **Alt**-Taste und **X** mit **Alt-X** bezeichnet.

- Steuerung über die **Ctrl**-Taste in Kombination mit einem Buchstaben.
Im folgenden wird die Kombination von Control-Taste und Taste **X** mit **Ctrl-X** abgekürzt.
- Anklicken des Bildschirms, wie zum Beispiel in Windows.
Die Menükürzel wurden so gewählt, daß sie leicht zu merken sind, die Cursortasten basieren auf dem Amiga-Basic-Editor, während die Control-Codes WordStar-kompatibel sind, wie bei vielen anderen Editoren auch. Wenn Sie mal nicht weiter wissen, bringt ein Druck auf die **Help**-Taste die Tastenbelegungen, falls sie nicht in den Menüs sichtbar sind.

2.2.2 Einige Worte über Requester

Der Editor macht extensiven Gebrauch von Requestern. Deshalb ist es ganz nützlich, zu rekapitulieren, wie man damit umgeht. Dies ist besonders für die Eingabe von Texten wichtig. Die Editor-Requester enthalten nur Text-Gadgets (Gadget = Bedienelement) und Knöpfe. Text-Gadgets erlauben es Ihnen, Text einzugeben und zu editieren. Sie erscheinen als Rechteck, das den Text enthält, und als Block, der die Cursor-Position anzeigt. Sie müssen dieses Rechteck anklicken, bevor Sie irgendein Zeichen eingeben können. Dann können Sie Zeichen eintippen und editieren, wozu auch die Cursor-Tasten, die **Backspace**- und die **Del**-Taste genutzt werden können. Sie können das ganze Feld mit **A-X** löschen. Sie können den Cursor mit **Shift** plus entsprechender Cursor-Taste an den Anfang oder das Ende des Feldes bewegen. Wenn sich mehr als ein editierbares Text-Gadget in einem Requester befindet, können Sie durch Anklicken mit der Maus hin- und herschalten.

Knöpfe können mit der Maus angeklickt werden, mit der Folge, daß der Requester verschwindet. Meistens gibt es einen Vorzugsknopf, der durch eine doppelte Einrahmung markiert ist. Wird **Return** gedrückt, hat das die gleiche Wirkung, als ob

Sie vorher ein Text-Gadget durch Klicken aktiviert hätten. In einige Requester können nur bestimmte Zeichen eingegeben werden. So akzeptiert der *Zeile* ...-Requester nur Ziffern.

2.2.3 Datei-Requester

Mit dem *Datei*-Requester können Sie Dateinamen für die Input- und Output-möglichkeiten von GenAm auswählen. Die einfachste Vorgehensweise besteht darin, zuerst die entsprechende Datei und dann *OK* anzuklicken. Um die gesamte Operation abubrechen, klicken Sie auf *Abbruch*. Am oberen Rand des Fensters befindet sich die »Schubladen«-Funktion, die anzeigt, welche Diskette und welches Verzeichnis angezeigt wird. Auch Wildcards sind zulässig für die Eingabe, z.B. zeigt

```
df1:examples/#?.s
```

alle Dateien an, die auf ».s« enden. Wenn Sie diese Spezifikation editieren wollen, geben Sie ein. Das Verzeichnis wird daraufhin gelesen, die entsprechenden Dateien werden im Hauptteil der Dialogbox angezeigt. Dateien werden durch einen Mausklick und anschließenden Klick auf ausgewählt. In der Dateiliste werden Verzeichnisse durch das Präfix »Dir« angezeigt. Mittels des Scroll-Balkens können Sie in der Liste blättern. Im vorliegenden Produkt wird die Datei-Dialogbox von Charlie Heath verwendet. Sollte diese Library nicht gefunden werden, wird statt dessen ein einfaches String-Gadget verwendet (wie auch schon in der Version 1.0). Mehr Details über die ARP-Library finden Sie im ARP-Unterverzeichnis auf Disk 2.

2.3 Eingabe von Text und Cursor-Bewegungen

Wenn Sie GenAm geladen haben, sehen Sie ein leeres Fenster mit einer Statuszeile am unteren Rand und einen den Cursor darstellenden Block in der oberen linken Ecke.

Die Statuszeile enthält Informationen über die Cursor-Position (Zeile und Spalte) sowie die Größe des noch freien Textspeichers in Bytes. Zu Anfang wird letztere 59980 Byte anzeigen, wenn die voreingestellte Größe von 60 Kbyte gewählt wurde. Sie können diesen Voreinstellungswert zusammen mit diversen anderen Optionen ändern, wenn Sie in *Voreinstellungen* gehen. Die »fehlenden« 20 Byte werden vom Editor intern belegt. Der Rest der Statuszeile wird für Fehlermeldungen gebraucht, die normalerweise mit einem Schirmflackern beginnen, um Sie darauf hinzuweisen. Jede Meldung wird entfernt, sobald Sie wieder eine Taste betätigen.

Die Texteingabe erfolgt über die Tastatur. Sobald Sie eine Taste drücken, wird das auf dem Schirm angezeigt und der Cursor um eine Spalte weiterbewegt. Wenn Sie sehr schnell tippen, kann es vorkommen, daß die Anzeige nicht folgen kann. Aber keine Sorge, es geht kein Tastendruck verloren, und die Anzeige erscheint, sobald Sie eine Pause machen. Am Ende einer Zeile können Sie die `[Return]`- oder `[Enter]`-Taste benutzen, um eine neue Zeile zu beginnen. Sie können Fehler mit der `[Backspace]`-Taste korrigieren, welche das Zeichen links vom Cursor löscht, oder mit der `[Del]`-Taste, die das Zeichen an der Cursor-Position löscht.

Der Hauptvorteil eines Computer-Editors gegenüber einer konventionellen Schreibmaschine ist die Fähigkeit, Dinge zu editieren, die Sie schon lange vorher eingegeben haben. GenAm bietet eine Fülle von Optionen, mittels derer Sie sich völlig frei im Text bewegen können.

2.3.1 Cursortasten

Um den Cursor durch den Text zu bewegen, um Fehler zu korrigieren oder um neue Zeichen einzugeben, benutzen Sie die vier Cursortasten. Wenn Sie den Cursor hinter das Zeilenende bewegen, fügt der Editor kein Leerzeichen hinzu. Wenn Sie dann ein Zeichen tippen, wird es allerdings automatisch an das wirkliche Ende der Zeile gehängt. Wenn Sie in sehr lange Zeilen Eingaben machen, rollt das Fenster nach rechts.

Wenn Sie mit `[↑]` über die erste Zeile des Fensters gehen, rollt der Text entweder nach unten (wenn weiter oben noch Text vorhanden ist), oder es erscheint die Meldung »Dateianfang«. Ähnlich wirkt `[↓]` am Ende des Textes, hier mit der Anzeige »Dateiende«. Für diejenigen unter Ihnen, die WordStar gewohnt sind: Die Tasten `[Ctrl]-[S]`, `[Ctrl]-[D]`, `[Ctrl]-[E]`, `[Ctrl]-[X]` wirken wie die Cursortasten.

Um den Cursor ein Wort weiter zu bewegen, kombinieren Sie die Tasten `[→]`/`[←]` mit der `[Shift]`-Taste. Ein Wort ist eine Zeichenkette, abgegrenzt von Leerzeichen, dem Tabulator-Zeichen oder an einem Ende durch Zeilenanfang oder Zeilenende. Sie können auch die WordStar-Codes `[Ctrl]-[A]` und `[Ctrl]-[F]` einsetzen. Um den Cursor eine Seite nach oben zu bewegen, nehmen Sie `[Shift]-[↑]` oder `[Ctrl]-[R]`. Eine Seite nach unten gehen Sie mit `[Shift]-[↓]` oder `[Ctrl]-[C]`. Wenn Sie den Cursor auf einen beliebigen Platz positionieren wollen, können Sie dafür den Mauszeiger nehmen und an dieser Stelle die linke Maustaste klicken. (Es gibt in WordStar kein Äquivalent für dieses Feature!)

2.3.2 Tabulator-Taste

Die Tabulator-Taste (**Tab**) schreibt ein spezielles Zeichen (ASCII-Code 9) in den Puffer. Auf dem Schirm erscheint eine Folge von Leerzeichen. **Tab** justiert den Cursor auf die nächste Spalte, die ein Vielfaches von 8 ist. Genauer gesagt: Wenn der Cursor am Beginn einer Zeile steht (Spalte 1) und Sie geben **Tab** ein, befindet sich der Cursor in Spalte 8+1, also 9. Tabs sind sehr nützlich, um Wörter untereinander zu justieren, weshalb ihre Hauptaufgabe in GenAm auch darin besteht, Befehle untereinander zu schreiben. Wenn Sie einen Tab löschen, rückt die Zeile auf, als ob die entsprechende Anzahl Leerstellen gelöscht worden wäre. Der Vorteil des Tabs ist, daß er nur ein Byte im Speicher belegt. Sie können die Tab-Breite ändern, bevor oder nachdem Sie GenAm laden.

2.3.3 Backspace-Taste

Die **Backspace**-(Rückschritt-)Taste löscht das Zeichen links vom Cursor. Wenn Sie **Backspace** am Anfang einer Zeile betätigen, löschen Sie das »unsichtbare« CR-Zeichen, die Zeile wird an das Ende der vorherigen angefügt. **Backspace** am Ende einer Zeile löscht das letzte Zeichen der Zeile, es sei denn, die Zeile ist leer. In diesem Fall wird der Cursor nur um ein Zeichen nach links verrückt.

2.3.4 Delete-Taste

Die **Del**-Taste löscht das Zeichen auf der Cursor-Position und hat keinen Effekt, wenn der Cursor hinter dem Ende einer Zeile steht.

2.3.5 Zeile ...

Um den Cursor auf eine bestimmte Zeile (unsichtbare Zeilennummer) zu bewegen, wählen Sie *Zeile...* aus dem Optionen-Menü oder geben **A**-**G** ein. Dann klicken Sie das darauf erscheinende Text-Gadget an und tippen die Zeilennummer, gefolgt von **Return**, ein. Sie können auch *OK* oder *Abbruch* anklicken, wenn Sie den Befehl zurücknehmen wollen. Nach der Auswahl wird der Cursor auf die entsprechende Zeile bewegt, die zugehörige Seite wird abgebildet. Wurde die Zeile nicht gefunden, erscheint die Meldung »Dateiende«.

2.3.6 Dateianfang

Um den Cursor auf die erste Zeile des Textes zu bewegen, wählen Sie *Dateianfang* aus dem Optionen-Menü oder geben **A**-**T** oder **Alt**-**↑** ein.

2.3.7 Dateiende

Um den Cursor auf die letzte Zeile des Textes zu bewegen, wählen Sie *Dateiende* aus dem *Optionen*-Menü oder geben **A-B** oder **Alt-↓** ein.

2.4 Verlassen von GenAm

Um GenAm zu verlassen, wählen Sie *GenAm verlassen* aus dem *Datei*-Menü oder geben **A-Q** ein. Falls Text ohne Sicherung auf Disk geändert wurde, erscheint eine Alertbox. Klicken Sie dort auf *Abbruch*, sind Sie wieder im Editor, bei *OK* hingegen verlassen Sie GenAm, die Änderung wird nicht gespeichert.

2.5 Löschen von Text

2.5.1 Löschen einer Zeile

Die aktuelle Zeile kann mit **Ctrl-Y** gelöscht werden. Wenn dies die letzte Zeile betrifft, wird das Fenster neu gezeichnet.

2.5.2 Löschen bis zum Ende einer Zeile

Der Text von der aktuellen Cursor-Position bis zum Ende der Zeile kann mit **Ctrl-Q** gelöscht werden. (Dies ist äquivalent zu **Ctrl-Q-Y** in WordStar.)

2.5.3 Gelöschte Zeile wieder einfügen

Wenn eine Zeile mit obigem Kommando gelöscht wird, wird sie in einem internen Puffer gespeichert und kann mit **Ctrl-U** wieder eingefügt werden. Dies kann mehrmals wiederholt werden und ist besonders nützlich, wenn Sie eine Zeile auf diese Art schnell kopieren wollen.

2.5.4 Gesamten Text löschen

Wenn Sie den ganzen Text löschen wollen, wählen Sie *Text löschen* aus dem *Datei*-Menü oder geben **A-C** ein. Wenn Sie im Text etwas geändert haben, das noch nicht auf Disk gesichert wurde, wird mittels eines Requesters eine Bestätigung verlangt. Klicken Sie auf *OK*, wenn Sie den Text tatsächlich löschen wollen, sonst auf *Abbruch*.

2.6 Disketten-Operationen

Es ist nutzlos, wenn Sie nur Text eintippen, ohne ihn danach abspeichern oder wieder laden zu können. Daher besitzt der Editor eine Menge Features, um auf Disk abzuspeichern, um ihn wieder laden zu können.

2.6.1 Text sichern

Um den Text, den Sie gerade editieren, zu sichern, wählen Sie *Sichern als* aus dem *Datei*-Menü oder geben **[A]-[S]** ein. Es erscheint ein Requester, der Ihnen die Eingabe eines passenden Dateinamens erlaubt. Wenn Sie *OK* anklicken oder **[Return]** drücken, wird die Datei auf Disk gesichert. Falls dabei ein Fehler auftritt, erscheint ein Requester mit einer Meldung oder einer AmigaDOS-Fehlernummer. Deren Bedeutung finden Sie in Anhang A. Wenn schon ein File gleichen Namens existiert, wird es gelöscht und durch den neuen Inhalt ersetzt. Wenn Sie im Requester *Abbruch* anklicken, wird das File nicht gesichert. Normalerweise wird eine Datei, die bereits existiert und den gleichen Namen trägt, gelöscht und durch die neue Version ersetzt. Wenn jedoch in den *Voreinstellungen Backups* gewählt wurde, wird ein bereits vorhandenes File umbenannt und erhält die neue Extension .BAK (wobei zugleich die alte BAK-Datei gelöscht wird).

2.6.2 Sichern

Wenn Sie bereits ein *Sichern als* (oder ein *Laden ...*) ausgeführt haben, merkt sich GenAm den File-Namen und zeigt ihn als Fenster-Titel an. Mit *Sichern* aus dem *Datei*-Menü oder **[Shift]-[A]-[S]** wird der alte Name einfach wieder eingesetzt. Wenn Sie versuchen, mit *Sichern* zu sichern, ohne vorher einen File-Namen definiert zu haben, erscheint das Requester wie unter *Sichern als* ...

2.6.3 Text laden

Um ein neues Text-File zu laden, wählen Sie *Laden ...* aus dem *Datei*-Menü oder geben **[A]-[L]** ein. Falls Sie das vorhandene File geändert haben, ohne es zu sichern, muß eine Bestätigung erfolgen. Der erscheinende File-Requester ermöglicht die Eingabe des File-Namens. Angenommen, Sie haben nicht *Abbruch* gewählt, wird der Editor versuchen, das File zu laden. Ist das File zu groß, erscheint die Meldung »Zuwenig Speicher«, und das Fenster blinkt kurz auf. Sie sollten dann, wie bereits geschildert, den Editor-Puffer größer wählen.

2.6.4 Text-File einfügen

Wenn Sie ein Text-File ab aktueller Cursor-Position einfügen wollen, wählen Sie *Datei einfügen* aus dem *Datei*-Menü oder geben **A-I** ein. Es erscheint der übliche File-Requester. Wenn Sie nicht auf *Abbruch* gehen, wird das File eingelesen, vorausgesetzt, es paßt noch in den Speicher.

2.6.5 Verzeichnis ändern

Durch dieses Kommando können Sie das aktuelle Verzeichnis ändern. Benutzen Sie einfach die Dialogbox, um in das gewünschte Verzeichnis zu gelangen.

2.7 Suchen und Ersetzen

Um einen bestimmten Text zu finden, wählen Sie *Suchen* aus dem *Suchen*-Menü oder geben **A-F** ein. Es erscheint ein Requester, der die Eingabe des Such-Textes und – wenn gewünscht – des Ersetz-Textes ermöglicht. Wenn Sie *Abbruch* anklicken, wird die Funktion abgebrochen. Ein Klick auf *Nächstes suchen* startet die Suche vorwärts, ein Klick auf *Vorheriges suchen* die Suche rückwärts. Wenn Sie nicht wollen, daß der gefundene Text durch einen anderen ersetzt wird, also gelöscht werden soll, lassen Sie das Ersetz-Feld leer. Wenn die Suche erfolgreich war, wird das Fenster neu gezeichnet und der Cursor auf den Anfang des Suchbegriffs gesetzt. Wenn der Text nicht gefunden werden konnte, erscheint die Meldung »Nicht gefunden« in der Statuszeile, und der Cursor bleibt an der gleichen Position. Bei der Suche sind Groß- und Kleinschreibung gleichwertig. Sie können also Test, test oder TEST eingeben.

Um das nächste Vorkommen des gesuchten Strings zu finden, wählen Sie *Nächstes suchen* aus dem *Suchen*-Menü oder geben **A-N** ein. Die Suche beginnt an der Position direkt nach dem Cursor. Um das vorherige Auftreten des Such-Textes zu finden, wählen Sie *Vorheriges suchen* aus dem *Suchen*-Menü oder geben **A-P** ein. Die Suche beginnt direkt ab der Cursor-Position. Wurde ein Such-Text gefunden, kann er durch den Ersetz-Text ersetzt werden, indem man *Ersetzen* im *Suchen*-Menü anklickt oder **A-R** eingibt. Nach dem Ersetzen sucht GenAm automatisch nach dem nächsten Auftreten des Such-Textes.

Wenn Sie jedes Vorkommen des gesuchten Textstrings ersetzen wollen, wählen Sie *Alle ersetzen* aus dem *Suchen*-Menü. Das Menü ist absichtlich so gestaltet, daß ein versehentliches globales Ersetzen verhindert wird. Aus diesem Grund gibt es dafür

auch kein Tastaturkürzel. Während des Vorgangs können Sie mit der **[Esc]**-Taste abbrechen. Tabs ersetzen Sie durch Drücken der **[Tab]**-Taste in der Dialogbox. Andere Control-Zeichen werden mittels direkter Eingabe gesucht (z.B. durch **[Ctrl-G]**). Dies funktioniert aber nicht bei CR (**[Ctrl-M]**) und bei LF (**[Ctrl-J]**).

2.8 Block-Kommandos

Ein Block ist ein markierter Textbereich, der kopiert, gelöscht, gedruckt oder auf Disk geschrieben werden kann. Dafür werden die Funktionstasten benutzt.

2.8.1 Markieren eines Blocks

Der Beginn eines Blocks wird markiert, indem man den Cursor auf die entsprechende Stelle bringt und **[F1]** drückt. Das Ende wird markiert, indem man an der entsprechenden Stelle **[F2]** drückt. Beginn und Ende eines Blocks können in beliebiger Reihenfolge markiert werden. Ein markierter Block wird hervorgehoben dargestellt. Wenn Sie innerhalb eines markierten Blocks eine Zeile editieren, wird sie erst dann hervorgehoben, wenn Sie sich aus der Zeile bewegen oder ein Kommando eingeben.

2.8.2 Sichern eines Blocks

Wenn ein Block markiert ist, kann er mit **[F3]** gesichert werden. Wenn kein Block markiert ist, erscheint die Meldung »Kein Block markiert«. Liegt der Anfang eines Blocks in der Textreihenfolge erst nach dem Ende, erscheint die Meldung »Ungültiger Block«. Beide Fehler brechen das Kommando ab. Wenn ein gültiger Block markiert ist, erscheint ein File-Requester für die Eingabe eines passenden Dateinamens. Wenn Sie einen Namen einer bereits existierenden Datei wählen, wird das vorhandene File überschrieben.

2.8.3 Kopieren eines Blocks

Ein markierter Block kann auf eine andere Textstelle kopiert werden, indem man den Cursor an die entsprechende Stelle bewegt und **[F4]** drückt. Wenn Sie versuchen, einen Block in sich selbst zu kopieren, erscheint die Meldung »Ungültiger Block«, und der Vorgang wird abgebrochen.

2.8.4 Löschen eines Blocks

Ein markierter Block kann mit **[Shift]-[F3]** oder **[Shift]-[F5]** gelöscht werden. Die Kombination mit der **[Shift]**-Taste wurde eingeführt, um ein versehentliches Löschen zu vermeiden. Ein gelöschter Block wird im Blockpuffer zwischengespeichert. Um Kompatibilität mit Devpac ST 2.0 und Devpac Amiga 1.0 zu gewährleisten, sind zwei Tastenkombinationen für diesen Befehl vorgesehen.

2.8.5 Block in den Blockspeicher kopieren

Mit **[Shift]-[F4]** wird ein markierter Block in den Blockspeicher kopiert. Diese Funktion ist dann sinnvoll, wenn Sie Textblöcke zwischen verschiedenen Dateien austauschen wollen. Dazu laden Sie zuerst eine Datei, kopieren einen Block in den Blockspeicher, laden dann die andere Datei und holen den Block wieder aus dem Blockspeicher.

2.8.6 Block einfügen

Durch Eingabe von **[F5]** wird ein Block aus dem Blockspeicher an der Cursor-Position eingefügt. Der Blockspeicher geht verloren, wenn die Textspeichergröße verändert wird oder assembliert wird.

2.8.7 Drucken eines Blocks

Ein markierter Block kann ausgedruckt werden, indem man *Block drucken* im *Datei*-Menü wählt oder **[A]-[W]** eingibt. Ein Requester erscheint, der nach dem Druckertyp fragt, wobei PRT: voreingestellt ist. Ein Klick auf *OK* startet den Ausdruck. Natürlich können Sie anstatt PRT: auch jedes andere gültige Peripheriegerät angeben, so daß Sie auch den Block in ein Disk-File drucken können. Der Unterschied zum Blockspeichern besteht darin, daß die Tab-Zeichen durch die entsprechende Anzahl Blanks ersetzt werden. Wenn kein Block markiert ist, wird das ganze File gedruckt.

2.9 Verschiedenes

2.9.1 Hilfsbildschirm

Die entsprechenden Tastenkombinationen für Befehle, die nicht im Menü angeführt werden, können durch Drücken der **[Help]**-Taste oder durch **[A]**-**[H]** am Bildschirm aufgelistet werden. In der erscheinenden Dialogbox werden auch der insgesamt noch freie Speicherplatz und die Versionsnummer von GenAm angegeben.

2.9.2 Voreinstellungen

Nach Auswählen von *Voreinstellungen* im *Optionen*-Menü erscheint eine Dialogbox auf dem Bildschirm, die es Ihnen ermöglicht, eine Anzahl von Editor-Einstellungen zu bestimmen:

Tabulatoren ändern

Die Voreinstellung für den Tabulator ist 8, kann aber während des Editierens durch die Wahl von *Tababstand* aus dem *Voreinstellungen*-Menü geändert werden. Es erscheint ein Requester, der die aktuelle Einstellung zeigt. Diese kann auf einen Wert zwischen 2 und 16 geändert werden.

Größe des Textpuffers

Die Grundeinstellung des Textpuffers beträgt 60 Kbyte, sie kann jedoch auf 4 Kbyte bis 999 Kbyte eingestellt werden, was die maximale Größe einer zu ladenden und editierenden Datei begrenzt. Sie sollten allerdings darauf achten, daß noch genügend Platz zum Speichern und für MonAm übrigbleibt. Eine Änderung des Arbeitsspeichers des Editors führt dazu, daß jeder Text, der gerade editiert wird, verlorengeht. Vorher wird jedoch eine Sicherheitsabfrage gemacht.

Backups

Per Grundeinstellung macht der Editor bei der Abspeicherung von Programmen keine Backups, dieses kann jedoch durch Klicken auf *Ja* in der *Voreinstellungen*-Box geändert werden.

Automatisches Einrücken

Es dient der Übersichtlichkeit beim Editieren von Programmen – sowohl in Assembler als auch in Hochsprachen –, Folgezeilen nach rechts einzurücken. Der Editor unterstützt dieses automatische Einrücken. Wenn es aktiviert ist, wird nach jedem **[Return]** zu Beginn einer neuen Zeile soviel an Leerraum (Tabs und Blanks) eingefügt, wie sich auch am Anfang der vorherigen Zeile befanden.

Zeilenende

Per Grundeinstellung bewegt sich der Cursor nicht nach links, wenn er sich bereits am linken Zeilenrand befindet. Ebenso kann er durch  nicht bewegt werden, wenn er am Ende einer Zeile steht. Durch Anklicken von *Umbruch* kann diese Voreinstellung dahingehend modifiziert werden, daß der Cursor am Zeilenende auch nach rechts wandern kann bzw. am Zeilenanfang nach links.

MonAm laden

Während der Initialisierung des Editors wird eine Kopie von MonAm automatisch geladen und ist per Tastendruck verfügbar. Sollte dies nicht erwünscht sein – Sie könnten ja auch dadurch 24 Kbyte Speicherplatz sparen –, kann die Voreinstellung durch diese Option geändert werden. Diese Änderung wird erst dann wirksam, wenn Sie die Voreinstellungen speichern und den Editor wieder aufrufen.

Volle Größe (autom.)

Per Grundeinstellung benutzt GenAm nicht den gesamten Bildschirm, dies kann aber durch Auswahl dieser Option dahingehend modifiziert werden, daß das Initialisierungsfenster so groß wie möglich erscheint.

Voreinstellungen speichern

Wenn Sie auf *OK* klicken, sind alle Voreinstellungen bis zum nächsten Aufruf des Editors wirksam. Wenn Sie die Konfiguration dauerhaft speichern wollen, klicken Sie *Sichern an*. Dadurch wird die Datei GENAM2.INF auf Disk gespeichert. Wenn Sie GenAm das nächste Mal aufrufen, wird die Konfiguration von dieser Datei eingelesen. Zusätzlich werden die Einstellungen der Assembler-Optionen abgespeichert.

2.10 Assemblieren und Programm ablaufen lassen

Alle Assemblier- und Ausführ-Optionen sind im Programm-Menü zu finden.

2.10.1 Assemblieren

Um das Programm zu assemblieren, das Sie gerade editieren, klicken Sie auf *Assemblieren ...* im *Programm*-Menü oder geben - ein. Die Bedeutung dieser verschiedenen Optionen und des Assemblierprozesses wird in Kapitel 3.2 beschrieben. Die einzige Option, die an dieser Stelle behandelt wird, ist die *Output*-Option.

GenAm kann auf *Diskette*, in den *Speicher* oder *Nirgendwohin* assemblieren – wenn Sie nur einen Syntaxcheck machen wollen, bietet sich die Option *Nirgendwohin* an, während Assemblieren in den Speicher ideal dafür ist, schnell etwas auszuprobieren.

Wenn Sie auf Disk assemblieren und Ihr Programm nicht unter einem bestimmten Namen abgespeichert haben, wird es automatisch unter NONAME gespeichert.

Nach dem Klicken auf *Assemblieren ...* oder Eingabe von **[Return]** beginnt der Assemblierungsprozeß. Am Ende der Assemblierung wartet das Programm auf einen Tastendruck und gibt Ihnen so die Möglichkeit, Meldungen zu lesen, bevor Sie in den Editor zurückkehren. Falls Fehler vorhanden sind, springt der Cursor in die erste fehlerhafte Zeile, die Fehlermeldung wird in der Statuszeile dargestellt. Nachfolgende Fehler (und Warnungen) werden nach Eingabe von **[A]-[J]** ausgegeben.

2.10.2 Ablauf von Programmen

Wenn Sie auf *Ausführen ...* im *Programm*-Menü klicken oder **[A]-[X]** (eXecute) eingeben, können Sie ein in den Speicher assembliertes Programm ablaufen lassen. Nach Beendigung des Programms kehren Sie in den Editor zurück. Wenn die Assemblierung nicht ordnungsgemäß beendet wurde, kann das Programm nicht ablaufen. Da Sie nach einem Absturz nicht mehr in den Editor zurückkehren können, empfiehlt es sich, den Source-Code vor dem Programmablauf zu sichern.

Warnung: Wenn während der Assemblierung nur nicht fatale Fehler wie z.B. undefinierte Symbole aufgetreten sind, können Sie das Programm zwar ablaufen lassen, jedoch kann nicht garantiert werden, daß dies ordnungsgemäß geschieht.

2.10.3 Debuggen

Wenn Sie ein zuvor in den Speicher assembliertes Programm debuggen wollen, klicken Sie auf *Debuggen* im *Programm*-Menü oder geben **[A]-[D]** ein. Dadurch startet der Debugger, durch Eingabe von **[Ctrl]-[C]** in MonAm wird das Debuggen wieder beendet.

Hinweis: Wenn die Option *MonAm* nicht eingeschaltet ist, kann diese Option nicht angewählt werden.

2.10.4 MonAm

Ein Klick auf *MonAm* im *Programm*-Menü oder die Eingabe von **[A]-[M]** ruft MonAm genauso auf, als ob es vom CLI aus durch Eintippen des Programmnamens aufgerufen worden wäre. Der Unterschied besteht jedoch darin, daß es sich in erster

Aufrufmöglichkeit bereits im Speicher befindet. Sie kehren in den Editor zurück, wenn Sie das Debuggen beenden.

Hinweis: Wenn die Option *MonAm* nicht eingeschaltet ist, kann diese Option nicht angewählt werden.

2.10.5 Anspringen der Fehlerzeile

Während des Assemblierens werden die ersten 30 Fehler und ihre Zeilennummern in einem Puffer gespeichert. Wenn Sie zum Editor zurückkehren, können Sie diese Zeilen schrittweise abarbeiten (ändern), indem Sie *Nächster Fehler* aus dem *Optionen*-Menü wählen oder **A**-**J** eingeben. Dies bewegt den Cursor auf die nächste Zeile, in der ein Fehler festgestellt wurde, und zeigt die Fehlermeldung in der Statuszeile an. Wenn Sie irgendwelche Änderungen vornehmen, die die Anzahl der Textzeilen ändern, wird der Fehlerpuffer gelöscht. Wenn keine Fehler mehr vorhanden sind, erscheint die Meldung »Welche Fehler!«.

2.11 Das Editor-Fenster

Das Fenster, das der Editor nutzt, funktioniert wie die meisten anderen Amiga-Fenster. So können Sie es mit dem Verschiebe-Gadget am oberen Rand verschieben. Sie können seine Größe mit dem Größen-Gadget links unten ändern. Sie können seine Lage relativ zu den anderen Fenstern ändern, indem Sie auf das Front-beziehungsweise Back-Gadget rechts oben klicken. Wenn Sie das Schließ-Gadget links oben anklicken, hat das denselben Effekt, als ob Sie *GenAm verlassen* aus dem *Datei*-Menü wählen oder **A**-**Q** eingeben.

Wie alle Intuitions-Fenster muß auch dieses vor einer Eingabe oder Menüwahl aktiviert werden (irgendwo im Fenster klicken). Wenn Sie einen größeren Bildschirmbereich als normal haben, wie zum Beispiel bei den europäischen PAL-Versionen (256 anstatt 200 Zeilen), können Sie das Fenster auf die volle Schirmgröße ziehen. Wenn GenAm während eines Diskzugriffs oder während des Assemblierens beschäftigt ist, erscheint anstelle des Mauszeigers eine Sanduhr, Menüpunkte sind licht dargestellt.

Kapitel

3

Der Makro-Assembler

3.1 Einführung

GenAm ist ein mächtiger, schneller Assembler, der mit dem Editor gekoppelt ist oder auch als alleinstehendes Programm aufgerufen werden kann. Er wandelt Text, der in den Editor eingetippt oder geladen wurde, zusammen mit optional von der Disk nachladbaren Texten in ein Binär-File um, wahlweise als linkbaren oder sofort ausführbaren Code.

3.2 Den Assembler aufrufen

3.2.1 Vom Editor aus

Sie können den Assembler vom Editor aus durch Wählen der *Assemblieren*-Funktion im *Programm*-Menü oder mit ☐A-☐A aufrufen. Eine Dialogbox erscheint:

Programmtyp	Ausführbar	Linkable
Groß/Klein	Verschieden	Gleich
Debug-Information	Nein	Normal Export.
Listing Nein	Bildschirm	Drucker Diskette
Assemblieren	Schnell	Langsam
Output	Nein	Speicher
Diskette:		
Assembler Optionen		
Abbruch	Assemblieren	

Programmtyp

Hier wählen Sie zwischen ausführbarem oder linkbarem Code. Die Unterschiede zwischen den Programmtypen werden später genau erklärt.

Groß/Klein

Hier können Sie wählen, ob in Symbolen Unterschiede in der Groß- oder Kleinschreibung gemacht werden sollen. Wenn Sie *Gleich* wählen, sind die Symbole »start« und »Start« gleichwertig; wenn *Verschieden* gewählt wird, werden sie als verschiedene Symbole behandelt.

Debug-Information

Wenn Sie Ihr Programm noch debuggen, sollten im fertigen Programm Symbole enthalten sein. Die Symboloptionen sind *Normal* und *Erweitert*: das HiSoft-erweiterte Debug-Format erlaubt es, bis zu 22 Zeichen lange Symbole anstatt der üblichen 8 Zeichen Länge zu verwenden.

Listing

Sie können hiermit die Ausgabe des Listings wählen: auf *Drucker* oder auf *Diskette*, auf der eine Datei mit gleichem Namen und der Extension .LST angelegt wird. Wenn Sie keine Ausgabe wollen, wählen Sie *Nein*.

Assemblieren

Hier können Sie die Geschwindigkeit des Assemblers bestimmen. Normalerweise sollten Sie den *Schnell*-Button betätigt lassen, wenn jedoch der Speicherplatz zu gering wird, sollten Sie *Langsam* wählen. Dadurch wird der Assembler gezwungen, so wenig Platz wie möglich zu belegen und den Zugriff auf Diskette zu verlangsamen.

Output

Wie bereits erwähnt, können Sie hier wählen, welches Programm aus Ihrem Quelltext erzeugt wird. *Nein* heißt, daß nur ein Syntaxcheck gemacht und kein Programm erzeugt wird. *Speicher* bedeutet, daß das Programm in den Speicher assembliert wird und so bereitsteht, um von MonAm debuggt oder auch einfach zur Ausführung gebracht zu werden. Es erfolgt kein Diskettenzugriff, außer wenn eine Include-Datei gebraucht wird, die meistens nur einmal gelesen wird. *Diskette* bedeutet, daß das Programm, wie gewohnt, auf Diskette erzeugt wird. Die Regeln der Namensgebung für die Dateien werden in Kürze erläutert. Wenn Sie alle gewünschten Optionen gewählt haben, klicken Sie auf *Assemblieren* oder drücken Return.

3.2.2 Der Stand-alone-Assembler

Format der Kommandozeile

Die Kommandozeile hat das Format

GenIm2 *Hauptdatei* [<-Optionen> [-Optionen]]

Hauptdatei ist der Name der Datei, die assembliert werden soll; wenn keine Extension angegeben wurde, wird .S angenommen. Optionen sollte ein »-« voranstellen. Erlaubte Optionen sind (zusammen mit den äquivalenten OPT-Direktiven):

- B keine Binärdatei erzeugen
- C Groß-/Kleinschreibung gleichwertig (OPT C-)
- D Debug (OPT D+)
- L linkbarer Code (OPT L+)
- M Langsam als Assembliermodus (Vorsicht: nicht identisch mit OPT M+)
- O Name der erzeugten Datei, folgt dem O ohne Leerschritt
- P Name der Listing-Datei, folgt dem P ohne Leerschritt
- Q In jedem Fall auf einen Tastendruck am Ende warten
- T Der Tab-Abstand; die Zahl folgt dem T ohne Leerschritt; z.B. benutzt -T10 einen Tab-Abstand von 10
- X Erweiterter Debug (OPT X+). Wird dann gebraucht, wenn bei der Generierung linkbaren Codes nur die exportierten Symbole exportiert werden sollen (OPT X+)

Die in GenIm2 verwendeten Optionen unterscheiden sich von denen der Version 1! Wenn keine Optionen in der Kommandozeile angegeben werden, wird eine ausführbare Programm-Datei erzeugt, deren Namen auf dem der Source-Datei basiert, allerdings ohne Listing und mit Unterscheidung zwischen Groß- und Kleinschreibung. Beispiele:

genim2 test -b	assembliert die Datei TEST.S, ohne eine binäre Ausgabedatei zu erzeugen; dies wäre ein Syntax-check, mehr nicht.
genim2 test -oram:test -p	assembliert TEST.S in die Datei RAM:TEST und erzeugt eine Listing-Datei namens TEST.LST.
genim2 test -ldprt:	assembliert TEST.S in linkbares Format mit vollständiger Debug-Information und schickt das Listing an die parallele Schnittstelle. Wenn Sie ein Listing an die serielle Schnittstelle senden wollen, müssen Sie AUX: angeben. Der Tab-Abstand des Listings beträgt 10 Zeichen.

Dateinamen

In GenAm ist geregelt, wie der Name einer Ausgabedatei kreiert wird. Der Name ist außerdem abhängig von den angegebenen Optionen, wie z.B. -O, und der Output-Direktive.

Wenn ein Dateiname explizit angegeben wird, ist

Name=angegebenerName

wenn die Output-Direktive nicht benutzt wurde, ist

Name=Source-Name + .PRG, .BIN oder .O

wenn die Output-Direktive eine Extension angibt, ist

Name=Source-Name + Extension vom Output

sonst ist

Name=Name vom Output

Der Assembliervorgang

GenIm ist ein Zwei-Paß-Assembler; während des ersten Passes wird der Quelltext im Speicher, ggf. auch von Diskette, verarbeitet und eine Symboltabelle gebildet. Wenn Syntaxfehler während des ersten Passes gefunden wurden, werden diese angegeben, der zweite Paß wird nicht gestartet. Während des zweiten Passes werden die mnemonischen Befehle in binäre Instruktionen umgewandelt, ein Listing kann ausgegeben – wenn erforderlich mit einer Symboltabelle – und eine Binärdatei erzeugt werden. Während des zweiten Passes gefundene Fehler und Warnungen werden angegeben. Während des Assemblierens kann jede Bildschirmausgabe durch Drücken einer beliebigen Taste angehalten und mit Return weiterlaufen gelassen werden. Mit Ctrl-C kann der Assembler abgebrochen werden; die bis dahin erzeugte Binär-Datei ist aber unvollständig und sollte nicht ausgeführt werden.

In den Speicher assemblieren

Um die Turn-around-Zeiten so gering wie möglich zu halten, kann GenAm in den Speicher assemblieren, so daß Ihr Programm sofort ausgeführt oder vom Editor sofort debuggt werden kann. Ein vom Speicher aus ablaufendes Programm verhält sich wie ein normales Programm und sollte lediglich mit einer RTS-Anweisung mit dem Stack, von dem aus in das Programm gegangen wurde, enden. Programme, die ein weiteres Mal ausgeführt wurden, modifizieren sich selbst und befinden sich in dem Zustand nach der letzten Ausführung. Wenn Sie *Voreinstellungen speichern* wählen, wird die aktuelle Assemblieroption zur Standardeinstellung gemacht.

3.3 Binäre Dateitypen

GenAm erzeugt zwei Dateitypen, die je nach Art der Applikation benötigt werden. Ihre Unterscheidung basiert auf der Extension des Dateinamens: Wenn keine Extension vorhanden ist, liegt eine direkt ausführbare Datei vor, wenn die Extension `.O` oder `.OBJ` lautet, liegt ein AmigaDOS-linkbares File vor.

Es gibt zwei Arten von ausführbaren Dateien: diejenigen, die vom CLI aus laufen, und die, die von der Workbench aus laufen. Der Unterschied liegt im Code, der ein Teil des Programms ist; der Assembler erkennt den Unterschied nicht. Anhang C.4 geht näher darauf ein. `.O`-Dateien können nicht sofort ablaufen, sondern müssen gemeinsam mit weiteren Sektionen in einen Linker eingelesen werden. Sie werden als linkbare Objektmodule bezeichnet.

3.4 Codearten

Wie in den meisten 16-Bit-Systemen wird unter AmigaDOS ein ausführbares Programm nicht an eine bestimmte Adresse geladen, sondern an eine solche Adresse, die von dem zu diesem Zeitpunkt freien Speicherplatz abhängig ist. Um das Problem der absoluten Adressierung in den Griff zu bekommen, enthält das Dateiformat Relozierinformationen, die AmigaDOS erlauben, das Programm nach dem Laden, jedoch vor dem Ablauf, zu relozieren. Zum Beispiel setzt der folgende Programmteil

```
move.l #string,a0
.
.
.
string dc.b 'Press any key',0
```

die absolute Adresse von »string« in ein Register, obwohl zur Assemblerzeit die Adresse von »string« nicht bekannt sein kann. Im allgemeinen kann der Programmierer Adressen auch dann als absolut behandeln, wenn er die wirklichen Adressen nicht kennt, während der Assembler (oder Linker) die notwendige Relozierinformation besorgt.

Hinweis: In manchen Programmen, Spielen oder für Entwicklungen auf mehreren Maschinen sind absolute Startadressen erforderlich. Aus diesem Grund wird die `ORG`-Direktive unterstützt.

3.5 Assembler-Befehlsformat

Jede Zeile, die der Assembler verarbeiten soll, muß folgendes Format besitzen:

Label-Feld	Mnemonic-Feld	Operanden-Feld	Kommentar-Feld
start	move.l	d0,(a0)+	Ergebnis speichern

Ausgenommen von dieser Regel sind Kommentarzeilen, die mit einem Stern (*) gekennzeichnet sind, und Leerzeilen, die ignoriert werden. Jedes Feld muß vom nächsten durch einen Zwischenraum getrennt sein, wofür jede Kombination von Leer- und Tab-Zeichen erlaubt ist.

3.5.1 Das Label-Feld (Markenfeld)

Ein Label (Marke) beginnt normalerweise in Spalte 1, aber wenn es nötig erscheint, es in einer anderen Spalte beginnen zu lassen, muß im direkten Anschluß ein Doppelpunkt (:) folgen. Labels sind grundsätzlich in allen Zeilen mit Befehlen erlaubt, jedoch bei einigen Assembler-Direktiven verboten, bei anderen wiederum sind sie Pflicht.

Alle Labels müssen mit einem der Zeichen a..z, A..Z, Umlauten oder dem Unterstrich (_) beginnen. Danach sind noch die Zeichen 0..9 und der Punkt erlaubt. Makro-Namen und Register-Equates dürfen keinen Punkt enthalten. Labels, die mit einem Punkt beginnen, und Zahlensequenzen, die mit einem \$ abgeschlossen sind, sind lokale Labels. Standardmäßig sind die ersten 127 Zeichen eines Labels signifikant, diese Anzahl kann jedoch reduziert werden. Labels sollten nicht die gleichen Namen wie Register tragen, ebenso nicht erlaubt sind die Namen der reservierten Wörter SR, CCR oder USP. Kleinbuchstaben werden normalerweise von den entsprechenden Großbuchstaben unterschieden, solange dies nicht über die Opt-C-Direktive verändert wird.

Einige Beispiele für korrekte Labels:

```
test, Test, TEST, _test, _test.end, test5, _5test
```

Beispiele für inkorrekte Labels:

```
.test, 5test, _&e, test>
```

In GenAm gibt es nur die reservierten Symbole LK, RS und G2, die mit dem Unterstrich beginnen.

3.5.2 Das Mnemonik-Feld

Das Mnemonik-Feld folgt dem Label-Feld und kann 68000-Befehle, Assembler-Direktiven oder Makro-Aufrufe enthalten. Einige Befehle und Direktiven erlauben die Typangabe mittels .B für Byte, .W für Wort und .L für Langwort sowie .S für Short. Was wann erlaubt ist, hängt vom jeweiligen Befehl beziehungsweise von der Direktive ab. GenAm ignoriert die Groß-/Kleinschrift-Unterscheidung bei Befehlen und Direktiven, weshalb move, MOVE und mOve als gleichwertig betrachtet werden.

3.5.3 Das Operanden-Feld

Für Befehle und Direktiven, die Operanden erfordern, enthält dieses Feld ein oder mehrere durch Kommas getrennte Parameter. Registernamen können groß oder klein geschrieben werden, das heißt, SP und sp sind identisch und gültige Namen für Stack-Zeiger.

3.5.4 Das Kommentar-Feld

Jedes Leer- oder Tab-Zeichen, das nicht in Anführungszeichen steht und das nach dem zu erwartenden Ende von Operand(en) gefunden wird, wird als Beginn des Kommentar-Feldes betrachtet, das der Assembler ignoriert.

Hinweis des Übersetzers: Vermeiden Sie unbedingt Leerstellen zwischen den Operanden, weil der Assembler dies als Kommentarbeginn auffaßt. Speziell bei Auflistungen nach DC kann das problematisch werden. Mindestens ein Leerzeichen (oder Tab) ist obligatorisch.

```
rts ;das ist OK
rts;das ist falsch
```

Beispiel für gültige Zeilen:

```
        move.l    d0,(a0)+      Kommentar hier
loop TST.W      d0
nur.label
    rts
* das ist eine reine Kommentar-Zeile
; eingerückt: link A6,#-10   schaffe Platz
a_string: dc.b 'Leerstellen in Hochkommas erlaubt' String
```

3.5.5 Ausdrücke

GenAm erlaubt komplexe Ausdrücke und unterstützt voll die Operatoren-Priorität, Klammern und logische Operatoren. Es gibt zwei Arten von Ausdrücken: absolute und relative. Dieser Unterschied ist wichtig. Absolute Ausdrücke sind Konstanten, die dem Assembler bekannt sind. Relative Ausdrücke sind (ergeben) Programmadressen, die während des Assemblierens nicht bekannt sind, weil das Programm vom AmigaDOS-Lader an eine beliebige Stelle im Speicher verlagert werden kann. Für einige Befehle und Direktiven gibt es Einschränkungen hinsichtlich des erlaubten Typs, einige Operatoren können nicht mit bestimmten Typ-Kombinationen benutzt werden.

Operatoren

Die zur Verfügung stehenden Operatoren – in absteigender Priorität – sind:

Monadisches Minus(–) und Plus(+) als Vorzeichen

Bitweises NOT (~)

Links (<<) und rechts(>>) schieben

Gleichheit(=); Größer als(>); Kleiner als(<)

Bitweises AND(&), OR(!) und XOR(^)

Multiplikation(*) und Division(/)

Addition(+) und Subtraktion(–)

Die Vergleichsoperatoren liefern als Wert 0, wenn der Ausdruck falsch ist bzw. –1 (\$FFFFFFF), wenn er wahr ist. Die Shift-Operatoren verschieben den links stehenden Operator um die Anzahl der im rechten Operator angegebenen Bits, die Leerstellen werden mit Nullen aufgefüllt. Die Priorität kann durch die Klammern () aufgehoben werden. Ausdrücke mit Operatoren gleicher Priorität werden von links nach rechts entwickelt. Leerstellen in Ausdrücken (ausgenommen in Strings in Hochkommas) sind nicht erlaubt, weil sie als Trennzeichen zum nächsten Feld aufgefaßt werden. Alle Ausdrücke werden in 32-Bit-Arithmetik mit Vorzeichen gerechnet, es findet kein Test auf Überlauf statt.

Zahlen

Absolute Zahlen sind in verschiedenen Darstellungen möglich als

Dezimale Konstanten, z.B. 1029

Hexadezimale Konstanten, z.B. \$12f

Oktale Konstanten, z.B. @730

Binäre Konstanten, z.B. %1000101

Zeichen-Konstanten, z.B. 'X'

\$ bezeichnet hexadezimale, % binäre, @ oktale Zahlen und ' oder " Zeichen-Konstanten.

Zeichen-Konstanten

Ob Anführungszeichen oder Hochkommas zur Stringabgrenzung benutzt werden, ist unbedeutend, jedoch müssen die Zeichen zu Beginn und Ende des Strings die gleichen sein. Zeichen-Konstanten können bis zu 4 Zeichen lang sein und werden rechtsbündig und mit führenden Nullen dargestellt, wenn erforderlich. Hier einige Beispiele mit den zugehörigen ASCII- und Hex-Äquivalenten:

```
"Q"      Q      $00000051
'hi'     hi     $00006869
"Test"   Test   $54637374
"it's"   it's   $6974277C
'it's'   it's   $6974277C
```

Strings in DC.B-Direktiven unterliegen leicht unterschiedlichen Regeln, was später noch gezeigt wird. Die in Ausdrücken verwendeten Symbole sind je nach Definition relativ oder absolut. Labels im Source sind relativ, wohingegen diejenigen, die mittels der EQU-Direktive verwendet wurden, dem Ausdruckstyp gleich sind, dem sie gleichgesetzt wurden. Ein Asterix (*) zeigt den Wert des Zählers am Programm-anfang an und ist immer relativ.

Erlaubte Typ-Kombinationen

Symbole in Ausdrücken werden entweder relativ oder absolut »entwickelt«, was davon abhängt, wie sie definiert wurden. Labels innerhalb des Programmbereichs (zu Befehlen) sind relativ, während solche auf EQU-Direktiven den Typ des zugewiesenen Wertes annehmen. Die folgende Tabelle faßt zusammen, welche Typ-Kombinationen zu welchem Resultat führen. Es bedeuten:

R Relativ
A Absolut
* Kombination ist nicht erlaubt, erzeugt Fehlermeldung.

	A op A	A op R	R op A	R op R
Shift operators	A	*	*	*
Bitwise operators	A	*	*	*
Multiply	A	*	*	*
Divide	A	*	*	*
Add	A	R	R	*
Subtract	A	*	R	A
Comparisons	A	*	*	A

Adressierungsarten

Die verfügbaren Adressierungsarten zeigt die folgende Tabelle. Bitte beachten Sie, daß GenAm beim Abarbeiten der Adressierungsarten Groß-/Kleinbuchstaben nicht unterscheidet, weshalb zum Beispiel sowohl D0 als auch a3 gültige Registernamen sind.

Form	Bedeutung	Beispiel
Dn	Datenregister direkt	D3
An	Adreßregister direkt	A5
(An)	Adreßregister indirekt (ARI)	(A1)
(An)+	ARI mit Postinkrement	(A5)+
-(An)	ARI mit Prädekrement	-(A0)
d(An)	ARI mit Offset	20(A7)
d(An,Rn.s)	ARI mit Index	4(A6,D4.L)
d.W	Absolut kurz	\$0410.W
d.L	Absolut lang	\$12000.L
d(PC)	PC-relativ mit Offset	NEXT(PC)
d(PC,Rn.s)	PC-relativ mit Offset und Index	NEXT(PC,A2.W)
#d	Konstante	

Hierin bedeuten:

- ARI: Adreßregister indirekt :
- n: Eine Zahl von 0 bis 7
- d: Eine Zahl
- R: Indexregister A oder D
- s: Typ W oder L; W, wenn nicht angegeben

Wenn Adreßregister indirekt mit Index verwendet werden, kann der Offset weggelassen werden. Zum Beispiel wird »move.l (a3,d2.1),d0« zu »move.l 0(a3,d2.1),d0«.

Spezielle Adressierungsarten

- CCR Condition Code Register (User-Byte des Statusregisters)
- SR Status-Register
- USP User Stack Pointer

Zusätzlich kann SP anstatt A7 in allen oben aufgeführten Adressierungsarten eingesetzt werden, z.B. 4(SP.D3.W). Daten- und Adreßregister können auch durch die

reservierten Symbole R0 bis R15 ausgedrückt werden. R0 und R7 sind äquivalent mit D0 bis D7, R8 bis R15 sind äquivalent mit A0 bis A7. Dies ist aus Kompatibilitätsgründen zu anderen Compilern der Fall.

3.5.6 Lokale Labels

GenAm unterstützt lokale Labels; das sind Labels, die einem bestimmten Teil des Sources angehören. Sie beginnen mit einem Punkt (».«) und sind vom vorhergehenden nicht-lokalen Label abhängig.

```
länge1 move.l 4(sp),a0
        .loop  tst.b  (a0)+
                bne.s  .loop
                rts

länge2 move.l 4(sp),a0
        .loop  tst.b  -(a0)
                bne.s  .loop
                rts
```

Es gibt zwei Labels namens »loop«, von denen das erste zu »länge1«, das zweite zu »länge2« gehört. Um Verwechslungen mit der absoluten Adressierungsart zu vermeiden, sind »w.« und »l.« nicht als lokale Labels zulässig. Wenn Sie globale Labels verwenden wollen, die mit einem Punkt beginnen, müssen Sie OPT U+ verwenden, und mit dem Unterstrich lokale Labels einführen.

Lokale Labels des Amiga-Makro-Assemblers

Ebenso können Strings von Zahlen, die mit »\$« abgeschlossen sind, als lokale Labels verwendet werden. Diese Eigenschaft besteht, um Kompatibilität mit anderen Assemblern zu gewährleisten. Wir empfehlen Ihnen die oben angegebene Form, um das Programm lesbar zu machen.

Symbole und Punkte

Symbole, die das Punkt-Zeichen enthalten, können in GenAm zu Problemen bei der absolut kurzen Adressierung und bei Register-Equates führen. Der Motorola-Standard zur Kennzeichnung der Kurzadressierung verursacht Probleme, weil Punkte auch in einem Label auftreten können, wie ein Beispiel am besten zeigt:

```
move.w    vector.w,d0
```

Dabei ist »vector« ein absoluter Wert, wie zum Beispiel eine Systemvariable. Da ein Label erwartet wird, wird die Fehlermeldung »undefined label« erzeugt. Um dies zu

vermeiden, sollte der Ausdruck, in diesem Fall ein Symbol, in Klammern gesetzt werden, wie z.B.

```
move.l (vector).w,d0
```

Der Punkt kann jedoch weiterhin nach numerischen Ausdrücken benutzt werden, wie z.B.

```
move.l $4.w,d0
```

Hinweis: GenAm1 und GenAm2 unterstützen in der Kurzwortadressierung anstatt des Punktes auch den »\«, allerdings sollte diese Adressierungsart aufgrund der Verwechslungsmöglichkeiten in Makroparametern vermieden werden.

3.6 Der Befehlssatz

3.6.1 Wortgrenzen

Alle Anweisungen außer DC.B und DS.B werden auf Wortgrenze assembliert. Falls Sie jedoch DC.B explizit auf einer Wortgrenze verwenden wollen, sollten Sie zuvor ein EVEN setzen. Obwohl alle Anweisungen, die es erfordern, auf Wortgrenze liegen, können Labels, denen nichts folgt, ungerade Werte haben und liegen nicht auf Wortgrenze. Dies wird am besten durch ein Beispiel gezeigt:

```
nop      ist immer auf Wortgrenze
dc.b 'ungerade'
start
tst.l (a0)+
bne.s start
```

Da »start« einen ungeraden Wert besitzt, würde hier das gewünschte Ergebnis nicht erzielt werden. Der Assembler gibt jedesmal, wenn bei einem BSR- oder BRA-Operanden eine ungerade Adresse gefunden wird, eine Fehlermeldung aus. Beachten Sie dabei, daß eine derartige Überprüfung nicht bei anderen Anweisungen stattfindet. Sie sollten also eine EVEN-Direktive voransetzen, wenn die Labels auf Wortgrenze liegen sollen. Hier unterlaufen leicht Fehler, da ja der vorhergehende String eine gerade Anzahl von Bytes lang ist. Wenn jedoch der String geändert wird, kann es Probleme geben.

Erweiterungen des Befehlssatzes

Der vollständige 68000-Befehlssatz wird von GenAm2 unterstützt. Es gibt aber einige stilistische Abkürzungen, die GenAm2 auch akzeptiert:

Condition Codes

Die alternativen Condition Codes HS und LQ, äquivalent mit CC und CS, werden bei Bcc-, DBcc- und Scc-Instruktionen unterstützt.

Branch-Befehle

Um einen kurzen Branch explizit anzugeben, benutzen Sie Bcc.B oder Bcc.S. Einen Branch mit Wort-Länge geben Sie mit Bcc.W an; Sie können aber auch den Optimierer die Branch-Länge wählen lassen. Bcc.L wird aus Kompatibilitätsgründen zu GenAml beibehalten, obwohl es genaugenommen ein 68020-Befehl ist; beim Auftreten eines Bcc.L wird eine Warnung generiert. Ein BRA.S zum darauffolgenden Befehl ist nicht erlaubt und wird, mit einer Warnung, in einen NOP verwandelt. Ein BSR.S zum darauffolgenden Befehl ist nicht erlaubt und erzeugt einen Fehler.

BTST

BTST ist einzigartig unter den Bit-Test-Befehlen, weil es PC-relative Adressiermodi unterstützt.

CLR

CLR An ist nicht erlaubt, benutzen Sie statt dessen SUB.L An,An; die Flags werden von diesem Befehl aber nicht beeinflusst.

CMP

Wenn der Source-Operand immediate ist, wird CMPI erzeugt, wenn der Ziel-Operand ein Adreßregister ist, wird CMPA benutzt. Wenn Sie die Adressierungsarten beider Operanden postinkrement benutzen, wird ein CMPM generiert.

DBcc

DBRA wird für DBF akzeptiert.

ILLEGAL

Das Op-Code-Wort \$4AFC wird generiert.

LINK

Wenn der Wert positiv oder ungerade ist, wird eine Warnung ausgegeben.

MOVE from CCR

Dies ist ein Befehl der Prozessoren 68010 und höher. Es wird in ein MOVE from SR konvertiert.

MOVEQ

Wenn der Wert im Bereich 128 bis 255 inklusive ist, wird eine Warnung ausgegeben. Wenn Sie ausdrücklich ein .L angeben, wird keine Warnung erzeugt.

3.7 Assembler-Direktiven

GenIm kennt bestimmte Pseudo-Mnemoniks. Diese Assembler-Direktiven, wie sie auch genannt werden, werden (normalerweise) nicht in 68000-Code übersetzt, sondern veranlassen den Assembler zu bestimmten Aktionen während des Assemblerlaufs. Diese Aktionen bewirken z.B. die Änderung des erstellten Objekt-Codes oder der Form des Listings. Direktiven werden genauso wie Befehle behandelt, können auch Labels haben (einige müssen Labels haben) und können von einem Kommentar gefolgt sein. Wenn Sie ein Label vor eine Direktive setzen, bei der das nicht unbedingt erforderlich ist, ist deren Wert undefiniert und wird normalerweise als Label ignoriert.

Im folgenden werden die Direktiven im einzelnen beschrieben. Bitte beachten Sie, daß die Schreibweise (groß/klein) unwichtig ist, im Handbuch wurde Großschreibung verwendet. Spitze Klammern (< >) bezeichnen optionale Parameter, Auslassungen (...) zeigen Wiederholungen an.

3.7.1 Assembler-Kontrolle

Die folgenden Direktiven beeinflussen den Assemblerlauf:

END

END bezeichnet das Textende für den Assembler, alles was danach kommt, wird ignoriert. END sollte am besten in der letzten Zeile stehen.

INCLUDE Dateiname

Diese mächtige Direktive holt ein Stück Quelltext von der Disk. Er wird genauso behandelt, als ob er im RAM stünde. Der Direktive muß ein Dateiname im gültigen AmigaDOS-Format folgen. Enthält der Name Leerstellen, muß er in Anführungs-

zeichen oder Hochkommas stehen. Laufwerks- und Directory-Angaben sind möglich, z.B.

```
include dfl:constants/header.s
```

Include-Direktiven können so tief geschachtelt werden, wie Speicherplatz vorhanden ist. Tritt ein Fehler beim Öffnen oder Lesen des Files auf, wird mit einer Fehlermeldung abgebrochen.

INCDIR "Directory"<,"Directory" u.s.w>

Diese Direktive gibt eine Liste von Directories an, in denen (und in dieser Reihenfolge) Include-Direktiven die Files suchen sollen. GenIm setzt dafür die Strings der Reihe nach vor die Filenamen der Include-Direktiven. Die INCDIR-Direktive muß vor Include-Direktiven im Text stehen.

INCBIN Dateiname

Mit dieser Direktive wird eine Datei vollständig in Ihr Programm mit einbezogen. Dies ist z.B. nützlich für Grafiken, die mit anderen Programmen erstellt wurden und von Ihrem Programm in binärer Form benutzt werden; es entfällt eine Konvertierung in DC-Direktiven.

OPT Option <,Option> ...

OPT erlaubt die Kontrolle diverser Optionen innerhalb von GenAm. Jede davon wird mit einem Buchstaben – gefolgt von einem Plus- oder Minuszeichen – bezeichnet. Mehrere Optionen können durch Kommas getrennt angegeben werden. Die folgenden Optionen (siehe auch OPT) sind erlaubt:

Option A – Automatische Adressierung im PC-Modus

Wenn OPT A+ spezifiziert ist, benutzt GenAm dort, wo es möglich ist, PC-Modus-Adressierung. Zum Beispiel wird

```
MOVE.L int_in,d0
```

```
in
```

```
MOVE.L int_in(pc),d0
```

bei der Assemblierung transformiert.

Der Gebrauch dieser Option kann eine erhebliche Reduzierung der Programmgröße und Laufzeit mit sich bringen, ohne daß dafür mehr eingegeben werden muß. Beachten Sie aber, daß dadurch nicht garantiert wird, daß positionsunabhängiger Code vorliegt. Wenn Sie den automatischen Gebrauch des PC-Modus überspringen wollen, sollten Sie den Ausdruck »(Ausdruck).L« ähnlich wie für Kurzworte

benutzen. Obiges Beispiel könnte also so manipuliert werden, daß absolute Adressierung verwendet wird:

```
MOVE.L(int_in).L,d0
```

Wenn keine solchen Transformationen durchgeführt werden, ist der Defaultwert A-.

Option C – Case-Unterscheidung bei Labels

Case-Sensitivität bedeutet, daß zwischen Groß- und Kleinbuchstaben unterschieden wird. Per Voreinstellung ist GenAm case-sensitiv, dies kann durch C- (nicht case-sensitiv) überschrieben oder mit C+ wieder zurückgesetzt werden. Die Signifikanz kann durch Angabe einer neuen Spezifikation zwischen dem C und dem jeweiligen Zeichen bestimmt werden. So gibt z.B. C16+ eine Signifikanz von 16 Zeichen an. Die Direktive muß im Text stehen, bevor irgendein Label eingesetzt oder definiert wird. Am sinnvollsten ist sie ganz am Anfang einer Source-Datei.

Option D – Debugging Information

Das AmigaDOS-Format für Binär-Files erlaubt es, eine Symboltabelle an das File-Ende zu hängen, die von Debuggern wie MonAm gelesen werden kann und beim Debuggen sehr nützlich ist. Per Voreinstellung ist diese Option nicht aktiv, kann aber mit D+ ein- bzw. mit D- ausgeschaltet werden. Es werden nur relative Labels aufgezeichnet, und zwar in Großschrift, wenn GenAm nicht auf case-sensitiv eingestellt ist. Wenn mehr als eine D-Option im Text auftritt, wirkt nur die letzte.

Option L – Linker Modus

Normalerweise erstellt GenAm ausführbaren Code, L+ generiert linkbaren Code, und L- stellt wieder ausführbaren Code ein. Die Direktive muß in der ersten Zeile des Textes stehen (auch keine Leerzeile davor!). Wenn der Linker-Modus aktiv ist, sind noch zusätzliche Direktiven möglich (wird an späterer Stelle noch gezeigt).

Option M – Makro-Expansion

Wenn ein Assembler-Listing erzeugt wird, werden darin enthaltene Makro-Aufrufe in der gleichen Form wie im Quelltext ausgegeben. Wenn Sie eine Auflistung der Befehle innerhalb der Makros wünschen, wählen Sie M+. M- schaltet diese Option wieder aus. Diese Direktive können Sie in einem Text so oft einsetzen wie erforderlich.

Option N – Narrow (schmales) Listing

Diese Option ist für Drucker mit nicht mehr als 80 Spalten vorgesehen. Wenn Sie mit N+ aktiviert ist, wird ein verkürztes Listing ohne Zeilennummern und ohne Objekt-Code erzeugt. Zum Umschalten auf »breit« dient N-.

Option O – Optimierung

GenAm besitzt die Fähigkeit, manche Befehle zu kürzen oder deren Geschwindigkeit zu erhöhen. Standardmäßig wird zwar keine Optimierung vorgenommen, jedoch kann jede Art von Optimierung wahlweise hinzugeschaltet werden.

- O1+ Optimiert Rückwärts-Branches wenn möglich, kann mit O1– wieder abgeschaltet werden.
- O2+ Optimiert Adreßregister indirekt mit Offset in Adreßregister, wenn der Offset Null ist; kann mit O2– abgeschaltet werden.
»move.l Wert(a0),d3« wird zu »move.l (a0),d3« wenn Wert gleich Null.
- O+ schaltet alle Optimierungen ein.
- O– schaltet alle Optimierungen ab.
- OW– schaltet die mit den Optimierungen verbundenen Warnungen aus, sie können mit OW+ wieder eingeschaltet werden.

Wenn während der Assemblierung eine Optimierung vorgenommen wurde, werden die Zahl der Optimierungen und die gespeicherten Bytes am Ende der Assemblierung angezeigt.

Option P – Prüfung auf positionsunabhängigen Code

Normalerweise erlaubt GenAm positionsunabhängigen Code. Wenn Sie aber nur PC-relativen Code generieren wollen, können Sie die Option P+ als Kontrolle einsetzen. Wenn der Code nicht positionsunabhängig sein sollte, werden Fehlermeldungen erzeugt. Mit P– wird die Kontrolle wieder ausgeschaltet. Mehrere P-Optionen in einem Text sind möglich, um zum Beispiel sicherzustellen, daß ein bestimmter Programmteil positionsunabhängig ist.

Option S – Symboltabelle

Wenn ein Listing erzeugt wird, gehört dazu eine Symboltabelle am Ende des Listings. Wenn Sie diese nicht benötigen, können Sie sie mit S– ausschalten bzw. mit S+ wieder einschalten. Gültigkeit besitzt nur die letzte Anweisung.

Anmerkung des Übersetzers: Die Symboltabelle der Include-Files kann sehr lang werden; es ist deshalb ratsam, sie abzuschalten.

Option T – Typen-Überprüfung

GenAm kann Programmierfehler entdecken, indem es Ausdrücke dahingehend prüft, ob sie absolut oder relativ sind. Da für manche Anwendungen und Programmierstile eine solche Meldung hinderlich sein kann, ist die Überprüfung mit T– aus- und mit T+ wieder einschaltbar.

```
main      bsr      init
          lea      main(a6),a0
          move.l(main).w,a0
```

Diese Zeilen würden normalerweise einen Fehler erzeugen, da »main« ein relativer Ausdruck ist und der Assembler in beiden Fällen einen absoluten Ausdruck erwartet. Falls aber der Code auf einer anderen 68000-Maschine laufen soll, kann dies zulässig sein: T- müßte also angegeben werden.

Option U – Unterstrich in lokalen Labels

Mit U+ können Sie angeben, daß lokale Labels mit »_« anstatt mit einem Punkt beginnen. Diese Option existiert, um Kompatibilität mit Compilern der Firma Prospero zu gewährleisten.

Option W – Warnungen

Um die Warnmeldungen von GenAm abzuschalten, geben Sie W- ein, W+ aktiviert sie wieder. Diese Direktive kann beliebig oft angewählt werden.

Option X – Erweiterter Debug

Dies ist eine erweiterte Version der Option D. Hiermit werden Symbole mit bis zu 22 Zeichen Länge anstatt maximal 8 Zeichen wie bei D ausgegeben.

3.7.2 Zusammenfassung der Optionen

- A Automatische PC-Modus-Adressierung
- C+ Case-Sensitivität und Signifikanz (C127+)
- C- Nicht case-sensitiv
- D+ Debugger-Information -> Binär-File
- D- Keine Debugger-Information -> Binär-File (Voreinstellung)
- L+ Erzeugt AmigaDOS-linkbaren Code
- L- Erzeugt ausführbaren Code (Voreinstellung)
- M+ Expandiert Makros im Listing
- M- Keine Makro-Expansion im Listing (Voreinstellung)
- N+ Schmales Listing
- N- Breites Listing (Voreinstellung)
- O+ Optimierung an
- O- Optimierung aus
- P+ Prüfung auf positionsunabhängigen Code
- P- Prüfung auf positionsabhängigen Code aus (Voreinstellung)
- S+ Symboltabelle mit Listing (durch LIST gesetzt)
- S- Keine Symboltabelle (durch NOLIST gesetzt)

T+ Typenüberprüfung an
 T- Typenüberprüfung aus
 W+ Warmmeldungen ausgeben
 W- Warmmeldungen unterdrücken
 X+ Erweiterter Debug

Zum Beispiel schaltet die Zeile

```
opt m+,s+,w-
```

die Makro-Expansion im Listing ein, veranlaßt die Ausgabe der Symboltabelle und unterdrückt Warmmeldungen.

<Label> EVEN

Diese Direktive setzt den Programmzähler auf eine gerade Adresse, d.h., er wird auf eine Wortgrenze justiert. Weil GenAm automatisch alle Befehle auf eine Wortgrenze justiert (ausgenommen DC.B und DS.B), ist EVEN nicht besonders oft erforderlich. Diese Direktive kann aber sehr nützlich sein, wenn man sicherstellen will, daß Strings und Puffer auf einer Wortgrenze beginnen.

CNOP Offset, Justierung

Diese Direktive justiert den Programmzähler nach dem Offsetwert und der Justierungsart. Eine Justierung von 2 heißt Wortgrenze, eine von 4 Langwortgrenze usw. Die Justierung ist relativ zum Programmstart, der immer auf eine Langwortgrenze gesetzt wird. Zum Beispiel:

```
cnop 1,4
```

justiert den Programmzähler auf die nächste Langwortgrenze plus 1 Byte. Beide Parameter müssen numerische, absolute Konstanten sein, Labels sind nicht erlaubt. Einige Teile des Amiga-Betriebssystems erfordern Langwortjustierung, was mit

```
cnop 0,4
```

erreicht werden kann. Bei Programmstart wird immer langwortjustiert.

<Label> DC.B Ausdruck<,>Ausdruck> ...

<Label> DC.W Ausdruck<,>Ausdruck> ...

<Label> DC.L Ausdruck<,>Ausdruck> ...

Diese Anleitungen definieren Konstanten im Speicher. Sie können einen oder mehrere Operanden, getrennt durch Kommas, besitzen. Die Konstanten werden auf Wortgrenzen für DC.W und DC.L gesetzt. In einer einzigen DC-Direktive können nicht mehr als 128 Byte erzeugt werden. DC.B behandelt Strings etwas anders als solche in normalen Ausdrücken. Die Regeln bezüglich der Anführungszeichen gelten auch hier, jedoch können Byte nicht gepaddet werden. Die String-Länge kann bis zu 128 Byte betragen.

Seien Sie vorsichtig mit Leerzeichen in DC-Direktiven, da ein Leerzeichen Kommentare einleitet! Die Zeile »dc.b 1,2,3 ,4« erzeugt nur 3 Byte, das vierte wird als Kommentar behandelt.

<Label> DS.B Ausdruck

<Label> DS.W Ausdruck

<Label> DS.L Ausdruck

Diese Direktiven reservieren Speicherplatz, die Inhalte werden auf Null gesetzt. Wenn ein Label enthalten ist, wird es auf den Anfang des definierten Bereichs gesetzt, der nur eine Wortgrenze für DS.W- und DS.L-Direktiven ist. Zum Beispiel reserviert jede der folgenden Zeilen auf verschiedene Weise 8 Byte Speicherplatz:

ds.b 8

ds.w 4

ds.l 2

<Label> DCB.B Zahl,Wert

<Label> DCB.W Zahl,Wert

<Label> DCB.L Zahl,Wert

Diese Direktive erlaubt, daß aus der spezifizierten Größe konstante Blöcke von Daten erzeugt werden. »Zahl« zeigt an, wie oft »Wert« wiederholt werden soll.

FAIL

Diese Direktive erzeugt den Fehler »user error«. Sie kann als Warnung für den Programmierer verwendet werden, wenn einem Makro eine falsche Anzahl von Parametern übergeben wurde.

OUTPUT Dateiname

Mit dieser Direktive läßt sich der Name der Ausgabedatei bestimmen. Wenn der angegebene Dateiname mit einem Punkt beginnt, wird nur die Extension der erzeugten Datei beeinflußt. Der Name wird sonst nach den oben beschriebenen Regeln gebildet.

__G2 (reserviertes Symbol)

Dieses Symbol kann mit der IFD-Bedingung verwendet werden, um festzustellen, ob mit GenAm2 assembliert wird. Der Wert dieses Symbols ist die Version des Assemblers und immer absolut.

Wiederholung von Schleifen

Manchmal ist es sehr nützlich, wenn eine oder mehrere Anweisungen mehrmals wiederholt werden können. Die Repeat-loop-Konstruktion erlaubt das.

<Label> REPT Ausdruck ENDR

Die zu wiederholenden Zeilen müssen innerhalb von REPT und ENDR stehen, die Anzahl der Wiederholungen steht im Ausdruck. Wenn der Ausdruck Null oder negativ ist, wird kein Code erzeugt. Wiederholungsschleifen können nicht verschachtelt werden. Ein Beispiel:

```
REPT    512/4    Schnelles Kopieren eines Sektors
move.l  (a0)+,(a1)+
ENDR
```

Hinweis: In Wiederholungsschleifen sollten keine Programmabels definiert werden, da sonst die Meldung »label defined twice« erscheint.

3.7.3 Listing-Kontrolle

Die folgenden Direktiven kontrollieren die Form des Listings:

LIST

Diese Direktive gibt das Listing im Pass 2 aus, und zwar auf das Gerät/File, das vorgewählt wurde, oder auf den Bildschirm. Alle folgenden Zeilen werden gelistet, bis eine END-Direktive, das Textende oder eine NOLIST-Direktive erreicht ist. Gibt es vorher keine NOLIST-Direktive, wird auch die Symboltabelle ausgegeben, es sei denn, diese wurde via OPT-S ausgeschaltet. Eine umfassendere Kontrolle über die Listing-Sektionen des Programms kann durch den Gebrauch der LIST+- oder LIST- -Direktive erreicht werden. Ein Zähler bestimmt, ob die Direktive an- oder ausgeschaltet ist. LIST+ addiert 1 zum Zähler, LIST- subtrahiert 1 vom Zähler. Wenn der Wert des Zählers Null oder positiv ist, ist LIST an-, wenn er negativ ist, ist LIST ausgeschaltet. Der Standard-Startwert beträgt -1 (also Listing aus). Besonders über Include-Dateien kann eine beträchtliche Kontrolle ausgeübt werden. Die normale LIST-Direktive setzt den Zähler auf 0, NOLIST setzt ihn auf -1.

NOLIST

NOLIST schaltet das Listing aus. Tritt danach keine LIST-Direktive mehr auf, wird auch keine Symboltabelle ausgegeben, es sei denn, daß dies über OPT-S vorgewählt wurde.

PLEN Ausdruck

PLEN setzt die Seitenlänge in Zeilen auf den Wert von »Ausdruck«. Voreingestellt ist 60, der neue Wert muß zwischen 12 und 255 liegen.

LLEN Ausdruck

LLEN setzt die Zeilenlänge in Zeichen auf den Wert von »Ausdruck«. Voreingestellt ist 132, der neue Wert muß zwischen 38 und 255 liegen.

TTL String

TTL druckt den Text-String, der in Hochkommas eingeschlossen sein kann (bei Leerstellen), als Titel zu Beginn jeder Seite. Wenn kein Titel spezifiziert wurde, wird der aktuelle Include-Dateiname verwendet.

SUBTTL String

Setzt den auf jeder Seite gedruckten Untertitel zum gegebenen »String«, der in Hochkommas stehen kann. Die erste Direktive setzt den Untertitel der ersten Seite usw.

SPC Ausdruck

SPC erzeugt so viele Leerzeilen im Listing, wie in »Ausdruck« angegeben.

PAGE

Page bewirkt einen Seitenvorschub im Listing.

LISTCHAR Ausdruck<,Ausdruck>...

Dies sendet die angegebene Zeichenfolge an das Listing-Gerät (mit Ausnahme des Bildschirms) und dient zum Beispiel zur Druckerumschaltung auf Schmalschrift. Wird zum Beispiel auf das Gerät PRT: ausgegeben, wird mit

```
listchar 27, '[' , '4' , 'w'
```

der Drucker in den Schmalschrift-Modus gesetzt. Wenn Sie PAR: oder SER: benutzen, müssen Sie die druckerspezifischen Codes senden. Zum Beispiel ergibt 15 den Schmalschrift-Modus auf Epson-Druckern und Kompatiblen.

FORMAT Parameter<,Parameter> ...

Durch diese Direktive ist eine exakte Kontrolle über das Format einer Zeile im Source-Code möglich. Jeder Parameter kontrolliert ein Feld im Listing und muß aus einer Zahl zwischen 0 und 2 bestehen, gefolgt von a+ oder a-:

- 0 Zeilennummer, dezimal
- 1 Sektionsname/-zahl und Programmzähler
- 2 Hexdaten in Worten bis zu 10 Worte, außer, wenn der Drucker breiter als 80 Zeichen ist.

3.7.4 Label-Direktiven

Die folgenden Direktiven sind zusammen mit Labels anzuwenden.

Label EQU Ausdruck

Equate heißt soviel wie Gleichsetzung, in einigen Assemblern wird dafür tatsächlich auch das Gleichheitszeichen (=) eingesetzt. Die EQU-Direktive setzt den Wert und den Typ von »Label« gleich dem Ergebnis von »Ausdruck«. Dieser darf keine Vorwärts-Referenzen und keine externen Labels enthalten. Bei einem Fehler in Ausdruck wird kein Wert zugewiesen. Das Label muß vorhanden sein.

label = Ausdruck

Die alternative Form des EQU-Ausdrucks.

Label EQU Register

Die EQU-Register-Direktive erlaubt es, ein Daten- oder Adreßregister mit einem vom Anwender vergebenen Namen anzusprechen, der als Label anzugeben ist. Dies nennt man Register-Equate. Ein Register-Equate muß vor dem ersten Gebrauch definiert werden.

Label SET Ausdruck

SET ist EQU ähnlich, allerdings mit dem Unterschied, daß mit SET (immer neuen SETs) einem Label andere Werte zugewiesen werden können. Vorwärts-Referenzen sind nicht erlaubt. SET ist besonders nützlich für Zähler in Makros, zum Beispiel:

```
zcount set zcount+1
```

wobei unterstellt wird, daß »zcount« am Anfang des Quelltextes auf 0 gesetzt wurde. Zu Beginn von Pass 2 werden alle SET-Labels wieder als undefiniert initialisiert, so daß ihre Werte in beiden Durchgängen (Pass 1 und Pass 2) dieselben sind.

Label REG Registerliste

Mit der REG-Direktive können Sie einem Symbol eine Registerliste zur Verwendung mit MOVEM zuordnen. Somit wird die Gefahr verringert, am Anfang und am Ende einer Routine unterschiedliche Registerlisten zu haben. Mit REG definierte Symbole können ausschließlich mit MOVEM benutzt werden.

<Label> RS.B Ausdruck

<Label> RS.W Ausdruck

<Label> RS.L Ausdruck

Mit diesen Direktiven können Sie eine Liste von konstanten Labels erstellen, was für Datenstrukturen und globale Variablen sehr nützlich sein kann. Zum näheren

Einstieg hier ein paar Beispiele: Nehmen wir an, Sie haben eine Datenstruktur vor sich, die aus einem Langwort, einem Byte und einem anderen Langwort besteht. Um Ihren Quelltext verständlicher zu machen, könnten Sie folgende Eingaben verwenden:

```
rsreset
d_next rs.l 1
d_flag rs.b 1
d_where rs.l 1
```

Sie können auf sie dann folgendermaßen zugreifen:

```
move.l d_next(a0),a1
move.l d_where(a0),a2
tst.b d_flag(a0)
```

Nehmen wir als weiteres Beispiel an, Sie weisen Ihren Variablen das Offset-Register A6 zu (wie in GenAm und MonAm). Sie könnten sie dann folgendermaßen definieren:

```
onstate rs.b 1
start rs.l 1
end rs.l 1
```

Sie können dann auf sie folgendermaßen zugreifen:

```
move.b onstate(a6),d1
move.l start(a6),d0
cmp.l end(a6),d0
```

Jede dieser Direktiven benutzt ihren eigenen Zähler, der bei Beginn jedes Passes auf 0 steht. Jedesmal, wenn der Assembler auf die Direktive trifft, wird das Label auf den aktuellen Wert gesetzt (mit Wortgrenze, wenn es .W oder .L ist). Es wird dann gemäß der Größe und Mächtigkeit der Direktive inkrementiert. Wenn obige Definitionen die ersten RS-Direktiven wären, würde »onstate« auf 0 stehen, »start« auf 2 und »end« auf 6.

RSRESET

Durch diese Direktive wird der interne Zähler – wie bei der RS-Direktive – zurückgesetzt.

RSSET

Dadurch kann der Zähler auf einen bestimmten Wert gesetzt werden.

RS (Reserviertes Symbol)

Dies ist ein reserviertes Symbol, das den aktuellen Wert des RS-Zählers beinhaltet.

3.7.5 Bedingtes Assemblieren

Bedingtes Assemblieren erlaubt es, ein Programm zu schreiben, das viele interne Optionen besitzt, zwischen denen während des Assemblierens ausgewählt werden kann. Das kann aus verschiedenen Gründen sehr nützlich sein:

- Ein Programm soll auf zwei verschiedenen Maschinen laufen.
- Debug-Code soll nur für die Testversion erzeugt werden.
- Zwei leicht unterschiedliche Versionen eines Programms werden gebraucht.

Wir selbst haben all diese Features während der Entwicklung des Devpac gebraucht. Der Assembler läuft auf zwei Maschinen, und nur in der hausinternen Version von GenAm gibt es Debug-Code.

Es gibt viele Direktiven, die zum bedingten Assemblieren eingesetzt werden können. Die Assemblierungsbedingungen können in Makros durch das Benutzen von Argumenten und durch die Definition von Symbolen in EQU- oder SET-Direktiven festgelegt werden. Zu Beginn eines bedingten Blocks muß eine der vielen IF-Direktiven stehen, an seinem Ende ein ENDC. Die bedingten Blöcke können bis zu 65535 Ebenen tief geschachtelt werden. Labels sollten nicht in den IF- oder ENDC-Zeilen stehen, weil diese Direktiven dann vom Assembler ignoriert werden.

IFEQ Ausdruck

IFNE Ausdruck

IFGT Ausdruck

IFGE Ausdruck

IFLT Ausdruck

IFLE Ausdruck

Diese Direktiven entwickeln den Ausdruck, vergleichen seinen Wert mit 0 und schalten in Abhängigkeit vom Ergebnis das bedingte Assemblieren an oder aus. Die Bedingungen selbst entsprechen den Bedingungs-Codes des 68000. Wenn zum Beispiel das Label DEBUG den Wert 1 hat, wirken die folgenden Anweisungen

```

           IFEQ  DEBUG
logon      dc.b  'Kommando eingeben: ',0
           ENDC
           IFNE  DEBUG
           opt   d+
logon      dc.b  'Wird's bald?',0
           ENDC

```

so: Die erste Bedingung schaltet das Assemblieren aus, weil 1 nicht gleich (not EQ) 0 ist, wohingegen die zweite Bedingung das Assemblieren einschaltet, weil 1 nicht gleich (NE) 0 ist. Beachten Sie, daß IFNE dem IF in Assemblern entspricht, die nur

eine Testbedingung kennen. Die Ausdrücke müssen zu korrekten Werten entwickelbar sein. Jeder Fehler hier gilt als fatal und führt zum Abbruch der Assemblierung.

IFD Label

IFND Label

Diese beiden Direktiven erlauben die Kontrolle in Abhängigkeit davon, ob ein Label definiert ist oder nicht. Mit IFD wird das Assemblieren eingeschaltet, wenn das Label definiert ist, wohingegen es mit IFND eingeschaltet wird, wenn das Label nicht definiert ist. Diese Direktiven sollten vorsichtig verwendet werden. Andernfalls kann unterschiedlicher Code in Paß 1 und Paß 2 erzeugt werden, was zu Phasenfehlern führt.

IFC 'String1','String2'

Diese beiden Direktiven vergleichen zwei Strings, die beide in Hochkommas stehen müssen. Sind beide gleich, wird das Assemblieren ein-, andernfalls wird es ausgeschaltet. Der Vergleich findet case-sensitiv statt (Groß-/Kleinbuchstaben werden unterschieden).

IFNC 'String1','String2'

Diese Direktive verhält sich ähnlich wie die vorhergehende, abgesehen davon, daß das Assemblieren eingeschaltet wird, wenn die Strings nicht gleich sind. Dies sieht dem ersten Anschein nach etwas nutzlos aus, aber wenn einer oder beide Parameter Makro-Parameter sind, kann dies sehr vorteilhaft sein.

ELSEIF

Durch diese Direktive wird das bedingte Assemblieren an- oder ausgeschaltet.

ENDC

Diese Direktive beendet die aktuelle Ebene des bedingten Assemblierens. Wenn es mehr IF als ENDC gibt, wird eine Fehlermeldung am Ende des Assemblerlaufs ausgegeben.

IIF Ausdruck Anweisung

Hier liegt ein Kurzausdruck der IFNE-Direktive vor, die die bedingte Assemblierung einer einzigen Anweisung oder Direktive erlaubt. ENDC sollte nicht zusammen mit IIF-Direktiven verwendet werden.

3.8 Makro-Operationen

GenAm unterstützt voll Makros im Motorola-Format. Das erlaubt zusammen mit der bedingten Assemblierung (oder auch allein), daß die Assembler-Programmierung wesentlich vereinfacht und die Lesbarkeit des Programms verbessert wird. Ein Makro ist ein Hilfsmittel für den Programmierer, der damit ganze Sequenzen von Direktiven spezifizieren kann, die sehr oft verwendet werden. Ein Makro wird zuerst definiert, damit sein Name in einem Makro-Aufruf, wie z.B einer Direktive mit bis zu 36 Parametern, verwendet werden kann.

Makro-Definitionen werden im Makro-Puffer gespeichert, welcher jeweils der freie Bereich am Ende des Textes ist, den Sie editieren. Die Anzahl freier Bytes – 4 in der Statuszeile – ist der zur Verfügung stehende Raum.

Label MACRO

Dies startet eine Makro-Operation und veranlaßt GenAm, alle folgenden Zeilen in den Makro-Puffer zu kopieren, bis ein ENDM auftritt. Makros können verschachtelt werden.

ENDM

ENDM beendet (nach einer MACRO-Direktive) das Abspeichern einer Makro-Definition. Ein Label ist in dieser Zeile nicht erlaubt.

MEXIT

Dadurch wird eine Makroentwicklung vorzeitig beendet und ist mit dem INC-Makro, das noch gezeigt wird, bestens erläutert.

NARG (reserviertes Symbol)

NARG ist keine Direktive, sondern ein reserviertes Symbol (eine Assembler-Variable). Ihr Wert ist die Anzahl der Parameter, die an das aktuelle Makro übergeben wurden, oder 0, wenn sich der Assembler nicht innerhalb eines Makros befindet. Wenn sich GenAm im case-sensitiven Modus befindet, ist die Schreibweise wichtig, NARG muß deshalb komplett in Großbuchstaben geschrieben werden.

Makro-Parameter

Wenn ein Makro mittels der MACRO-Direktive definiert wurde, kann es aufgerufen werden, indem man seinen Namen, gefolgt von bis zu 36 Parametern, als Direktive verwendet. Im Makro selbst stehen die Parameter als Backslash-Zeichen (\), gefolgt von einer Ziffer von 1 bis 9. Sie werden durch die jeweiligen Aufrufparameter

ersetzt (oder auch nicht, wenn sie fehlen). Der spezielle Parameter \0 bewirkt, daß der Typ (B, W, L) beibehalten wird, der dem Makro beim Aufruf eventuell angehängt wurde (wird W, wenn der Typ fehlt). Durch die Verwendung der Syntax »\<Symbol>« oder »\\${<Symbol>« kann ein Symbol in eine hexadezimale oder dezimale Abfolge von Zahlen umgesetzt werden. Das Symbol muß zum Zeitpunkt der Expansion definiert sein.

Der Parameter »\@« dient dazu, unterschiedliche Labels bei jedem Makro-Aufruf zu erzeugen. »\@« wird in der Makro-Entwicklung durch »_nnn« ersetzt, wobei nnn eine Zahl ist, die bei jedem Aufruf um 1 erhöht wird. Wenn ein Makro-Parameter Leerstellen oder Kommas enthält, sollte er in spitze Klammern (< >) gesetzt werden. Per Voreinstellung wird im Assemblerlisting nur der Makro-Aufruf und nicht der produzierte Code gezeigt. Jedoch kann mit der schon beschriebenen OPT-M-Direktive die Makro-Erweiterung mit Listing an- und abgeschaltet werden. Ein echtes »\« muß in einer Makro-Definition durch »\\« spezifiziert werden.

Ein Makro-Aufruf kann über mehr als eine Zeile gehen, was besonders bei Makros mit einer langen Liste von Parametern erforderlich ist. Der Makro-Aufruf muß dafür mit einem Komma enden, die nächste Zeile muß mit einem von Tabs oder Leerzeichen gefolgt »&« beginnen.

Im Assemblerlisting zeigt der Default-Wert nur den Makro-Aufruf und nicht den erzeugten Code. Makro-Expansion-Listings können durch die OPT-M-Direktive an- und ausgeschaltet werden. Makro-Aufrufe können so tief geschachtelt werden, wie Speicherplatz vorhanden ist, und können auch rekursiv sein. Makro-Namen werden in einer separaten Symboltabelle gespeichert, so daß es keine Überschneidungen mit ähnlich benannten Routinen geben kann. Sie können mit einem Punkt beginnen.

Makro-Beispiele

Beispiel 1: Aufruf des BDOS

Im allgemeinen werden Amiga-Library-Routinen so aufgerufen:

Register A6 sichern

A6 mit dem Library-Zeiger laden

ein JSR mit Offset(A6) ausführen

A6 zurückspeichern

Ein Makro, das die oben genannte Folge abarbeitet, könnte lauten

```
call_lib MACRO
    move.l    a6,-(sp)
    move.l    \2,a6           Hole Lib-Zeiger
    jsr       _LV0\1(a6)     Lib-Routine aufrufen
    move.l    (sp)+,a6        A6 zurück
ENDM
```

Die Direktiven sind nur in Großschrift, um sie hervorzuheben. Diese Schreibweise ist nicht zwingend erforderlich. Wenn Sie dieses Makro nutzen wollen, um die DOS-Funktion Output aufzurufen, gilt:

```
call_lib Output, _DOSBase
```

Wenn das Makro expandiert wird, wird \1 durch Output und \2 durch _DOSBase ersetzt. \0 – falls im Makro benutzt – würde W sein, da kein Typ im Aufruf angegeben wurde. Der oben genannte Aufruf wird assembliert als

```
move.l    a6, -(sp)
move.l    \2, a6           Hole Lib-Zeiger
jsr       _LV0\1(a6)       Lib-Routine aufrufen
move.l    (sp)+, a6        A6 zurück
```

Beispiel 2: Ein INC-Befehl

Der 68000 hat keinen INC-Befehl wie andere Prozessoren, derselbe Effekt kann aber auch mit einer ADDQ #1-Anweisung erreicht werden. Folgendes Makro kann dazu benutzt werden:

```
inc MACRO
    IFC ' ', '\1'
    fail                      Fehlender Parameter
MEXIT
ENDC
addq.\0 #1, \1
ENDM
```

Ein Beispiel-Aufruf wäre

```
inc.l    a0
```

dies wird expandiert zu

```
addq.l    #1, a0
```

Das Makro beginnt mit dem Vergleich des ersten Parameters mit einem leeren String. Wenn Gleichheit besteht, wird über FAIL eine Fehlermeldung verursacht. Die MEXIT-Direktive wird eingesetzt, um das Makro zu verlassen, ohne daß der Rest expandiert wird. Wenn kein leerer Parameter existiert, wird die ADDQ-Anweisung ausgeführt, wobei der Typ des \0-Parameters eingesetzt wird.

Beispiel 3: Ein Makro zur Lösung von n-Fakultät

Obwohl man praktisch das Problem nicht so löst, definiert dieses Makro ein Label als Fakultät einer Zahl. Dies zeigt aber sehr schön, wie Rekursionen in Makros funktionieren können. Bevor das Makro gezeigt wird, ist es nützlich, sich anzusehen, wie die Lösung in einer Hochsprache wie Pascal aussehen würde:

```

function factor(n:integer):integer;
begin
    if n>0 then
        factor:=n*(factor-1)
    else
        factor:=1
    end;

```

Die Makro-Definition nutzt die SET-Direktive für Multiplikationen wie $n*(n-1)*(n-2)$ usw. auf diese Art:

```

factor      MACRO
            IFND      \1
\1          set      1          setze 1 für ersten Aufruf
            ENDC
            IFGT      \2
\1          factor   \1,\2-1     nächste Ebenen abarbeiten
            set      \1*(\2)     n=n*(factor-1)
            ENDC
            ENDM

```

```

* Beispielaufruf
    factor      test,3

```

Das Ergebnis ist, daß die Variable »test« auf 3! (Fakultät von 3) gesetzt wird. Der Grund dafür, daß beim zweiten SET (\2) anstatt \2 geschrieben wurde, ist, daß dieser Parameter normalerweise nicht ein simpler Ausdruck ist, sondern eine Liste von Zahlen, die durch Minuszeichen getrennt sind. Das könnte also so assembliert werden:

```

test      set      test*5-1-1-1
(d.h. test*5-3) anstatt des richtigen
          set      test*(5-1-1-1)
(d.h. test*2)

```

Beispiel 4: Bedingtes Return

Der 68000-CPU fehlt das bedingte Return, wie man es bei anderen Prozessoren findet. Hier kann jedoch dafür ein Makro definiert werden, das den \@-Parameter benutzt. Zum Beispiel könnte ein Makro für »return if EQ« so aussehen:

```

rtseq      MACRO
            bne.s     x\@
            rts
x\@
            ENDM

```

Der \@-Parameter wurde genommen, um unterschiedliche Labels bei jedem Aufruf zu generieren. So würden in diesem Fall Labels wie x_002 oder x_017 entstehen.

Beispiel 5: Numerische Substitution

```

vname      MACRO
            dc.b      \1,\<version>,0
            ENDM
            .
            .
version    equ        42
            vname    <'Buchstabensuchprogramm v'>

wird expandiert zu:
            dc.b      'Buchstabensuchprogramm v','42',0

```

Beispiel 6: Ein komplexer Makro-Aufruf

Nehmen wir an, daß ein Programm eine komplizierte Tabellenstruktur aufweist, die eine variable Anzahl von Feldern beinhaltet. Ein Makro kann z.B. auch dafür geschrieben werden, daß angegebene Parameter verwendet werden:

```

Tabellen_Eintrag    MACRO
                    dc.b      .end\@-*           Längenbyte
                    dc.b      \1                 immer
                    IFNC      '\2',''
                    dc.w      \2,\3             2. und 3.
                    ENDC
                    dc.l      \4,\5,\6,\7
                    IFNC      '\8',''
                    dc.b      '\8'             text
                    ENDC
                    dc.b      \9
                    .end\@
                    dc.b      0
                    ENDM

```

* Beispiel für einen Aufruf

```

                    Tabellen_Eintrag    $42,,,t1,t2,t3,t4,
&                    <namen Eingeben:>,%0110

```

Dieses Beispiel zeigt sehr deutlich, daß Makros die Programmierarbeit enorm erleichtern können. In diesem Fall wird mit einer Datenstruktur gearbeitet, die aus einem Längenbyte (im Makro mit Hilfe von \@ errechnet), zwei optionalen Worten, vier Langworten, einem optionalen String, einem Byte und einem Null-Byte besteht. Der Code, der aus diesen Beispiel resultiert, sieht so aus:

	dc.b	.end_001
	dc.b	\$42
	dc.l	t1,t2,t3,t4
	dc.b	'Namen eingeben:'
	dc.b	%0110
.end_001	dc.b	0

3.9 Ausgabedatei-Formate

GenAm kann sowohl ausführbaren als auch linkbaren Code erzeugen. Manche Direktiven erfordern verschiedene Aktionen, die abhängig davon sind, welches Format der Ausgabedatei gewählt wird. Details darüber, wie jedes Format verwendet wird, werden im folgenden gegeben.

Ausführbare Dateien

Diese sind direkt ausführbar, z.B. durch Doppelklicken von der Workbench aus, oder durch Eingabe des Namens im CLI. Die Datei kann verschiedene Sektionen, Relozierinformation und/oder Symbolinformationen enthalten. Diese Dateien besitzen normalerweise keine Extension.

Linkbare Dateien

Wenn Sie größere Programme oder Module in Assembler-Sprache schreiben, die von einer High-Level-Sprache verwendet werden sollen, müssen Sie eine linkbare Datei erzeugen. Das AmigaDOS-Linkerformat wird von den meisten High-Level-Sprachen für den Amiga unterstützt. Normalerweise besitzen sie die Extension .O oder .OBJ.

3.9.1 Das richtige Format

Wenn Sie mit einer Hochsprache zusammenarbeiten, haben Sie meistens keine Wahl: Sie müssen das vom Compiler unterstützte Format verwenden.

Wenn Sie ausschließlich in Assembler schreiben, werden Sie wohl das ausführbare Format benutzen: die Assemblierzeit ist gering, Linken fällt weg, und Sie können direkt in den Speicher assemblieren, um die Turn-around-Zeiten so klein wie möglich zu halten. Wenn Sie ein größeres Programm schreiben, womöglich im Team, sollten Sie allerdings linkbaren Code bevorzugen.

3.10 Ausgabedatei-Direktiven

In diesem Abschnitt werden diejenigen Direktiven beschrieben, deren Wirkung vom Ausgabedatei-Format abhängig ist. Das richtige Format kann durch folgende Methoden gewählt werden: durch die Eingabe von Kommandozeilen, wenn GenIm direkt vom CLI aus abläuft, durch Anklicken der entsprechenden Optionen in der Assemblieren-Optionen-Dialogbox im Editor oder mit der OPT-L-Direktive am Anfang der Quelldatei.

3.10.1 Sektionen

SECTION String<,Typ>

Diese Direktive verursacht einen Wechsel zu der genannten Sektion, der Typ wird optional definiert. Ein Programm kann aus mehreren Sektionen bestehen, die im endgültigen Programm mit gleichnamigen Sektionen zusammengefügt werden. »String« und »Typ« werden als Sektionsname verwendet. Sie können jeweils bis zu 32 Zeichen lang sein, dürfen nicht in Klammern stehen und keine Tabs oder Leerzeichen enthalten. Die Schreibweise des Namens ist signifikant. In einem Programm können mehrere SECTION-Direktiven stehen. Es gibt folgende Typen:

CODE	Codesektion, Public Memory
CODE_F	Codesektion, Fast Memory
CODE_C	Codesektion, Chip Memory
DATA	Datasektion, Public Memory
DATA_F	Datasektion, Fast Memory
DATA_C	Datasektion, Chip Memory
BSS	BSS-Sektion, Public Memory
BSS_F	BSS-Sektion, Fast Memory
BSS_C	BSS-Sektion, Chip-Memory

CODE-Sektionen werden für ausführbare Anweisungen gebraucht, DATA-Sektionen für initialisierte Daten (Konstanten) und BSS für nichtinitialisierte Daten. BSS-Sektionen besitzen den Vorteil, daß sie keinen Speicherplatz einnehmen – nur die Länge der BSS-Sektion wird gespeichert. Wenn Sie eine Sektion als BSS definieren, können Sie nur mit der DS-Direktive ausführbaren Code erzeugen – jede andere Anweisung würde einen I/O-Fehler in der Binärdatei erzeugen.

Hinweis: Verwenden Sie in Versionen des Linkers ALINK vor 2.3 keine Typen, die Fast Memory oder Chip Memory erfordern, da es sonst zum Absturz kommt.

IDNT string

Dies setzt den Namen »string« als Modulname ein. Er kann bis 32 Zeichen lang sein, sollte keine Tab- und Leerzeichen enthalten und nicht in Anführungszeichen geschrieben werden. Nur eine IDNT-Direktive ist pro Modul zulässig.

3.10.2 Importe und Exporte

Bei beiden linkbaren Formaten ist es wichtig, Symbole importieren und exportieren zu können, was sowohl für relative Symbole wie Programmabels als auch für absolute Symbole wie Konstanten gilt. Das AmigaDOS-Linker-Format macht keinen Unterschied zwischen diesen beiden Typen; der Assembler kann jedoch eine Typenüberprüfung vornehmen, wenn die Typen beim Import spezifiziert wurden, und dadurch Fehler im Programm entdecken, die sonst vielleicht übersehen worden wären.

XDEF Export<,Export>...

Dadurch werden Labels für den Export in andere Programme oder Module definiert. Wenn eines der spezifizierten Labels nicht definiert ist, erscheint eine Fehlermeldung. Lokale Labels können nicht exportiert werden.

Ausführbar: Diese Direktive wird ignoriert.

XREF Import<,Import>...**XREF.L Import<,Import>...**

Dadurch werden Labels als externe Referenzen definiert, d.h., sie werden erst im Linkerlauf von anderen Programmen oder Modulen importiert und sind im Assemblerlauf unbekannt. Mit XREF werden relative Labels, d.h. Programmadressen, definiert, wohingegen mit XREF.L absolute Labels, d.h. Konstanten, definiert werden.

Ausführbar: Diese Direktive wird ignoriert.

Importe in Ausdrücken verwenden

Ausführbar: Es gibt keine Importe! Referenzen zu Labeln in anderen Sektionen unterliegen den unten beschriebenen Einschränkungen.

Linkbar: Importe können in Ausdrücken verwendet werden, jedoch ist nur ein Import pro Ausdruck erlaubt. Ein Ausdruck mit einem Import muß die Form `Import + Zahl` oder `Import - Zahl` haben. Importe können mit beliebig komplexen Ausdrücken kombiniert werden, jedoch muß der komplexe Ausdruck vorangestellt werden, wie z.B. in

```
move.l 3+(1<<count+5)+import
```


Ausdruck	Import	Intersektion	Beispiel
Byte	J	N	move.b #test,d0
Wort	J	J	move.w test(a3),d0
Langwort	J	J	move.l test,d0

COMMENT Kommentar

Diese Direktive wird ignoriert.

ORG Ausdruck

Diese Direktive läßt den Assembler positionsabhängigen Code erzeugen; der PC wird auf den angegebenen Ausdruck gesetzt. Normalerweise brauchen AmigaDOS-Programme kein ORG, auch wenn sie positionsunabhängig sind. Die ORG-Direktive erlaubt die Erzeugung von ROM-Code oder Code, der für andere 68000-Rechner gedacht ist. Es darf mehr als ein ORG im Programm stehen, es wird aber kein »Padding« vorgenommen.

Ausführbar: Diese Direktive sollte mit höchster Vorsicht gehandhabt werden, da das erzeugte Programm wahrscheinlich nicht problemlos ausführbar ist. Am Anfang der erzeugten Datei steht zwar der AmigaDOS-Header, jedoch am Ende keine Relozierinformation.

Linkbar: Diese Direktive ist nicht erlaubt.

OFFSET <Ausdruck>

Hiermit wird die Code-Generierung in eine besondere, assembler-interne Sektion umgeschaltet. Der Ausdruck, der nicht angegeben werden muß, setzt den PC dieser Sektion. Es werden keine Daten auf Diskette geschrieben, innerhalb dieser Sektion ist nur die DS-Direktive erlaubt. Labels, die innerhalb dieser Sektion definiert werden, sind absolut. Um einige Systemvariablen des ST (!) zu definieren, sähe der Quelltext so aus:

```

OFFSET      $400
etv_timer           ds.l      1      soll $400 betragen
etv_critic          ds.l      1      404
etv_term            ds.l      1      408
ext_extra           ds.l      5      40C
memvalid            ds.l      1      420
memcntl             ds.w      1      424

```

__LK (reserviertes Symbol)

Dies ist ein reserviertes Symbol. Es kann dazu benutzt werden, zu erkennen, welche Code-Art erzeugt wird. Der Wert dieses Symbols ist immer absolut und kann eine der folgenden Größen haben:

- 3 linkbar
- 4 ausführbar

3.11 Zusammenfassung der Direktiven

Assemblerkontrolle

END	Ende Quelltext
INCLUDE	Liest Quelltext von Disk
INCDIR	Setzt Include-Directory
INCBIN	Liest Binärdatei von Diskette
OPT	Optionskontrolle
EVEN	PC auf gerade Adresse
CNOP	Justiert PC
DC	Definiert Konstanten
DS	Definiert Leerzeichen
DCB	Definiert Konstantenblock
FAIL	Erzwingt Fehlermeldung

Schleifenwiederholung

REPT	Beginn des zu wiederholenden Blocks
ENDR	Ende des zu wiederholenden Blocks

Listingkontrolle

LIST	Listing an
NOLIST	Listing aus
LLEN	Seitenlänge setzen
PLEN	Zeilenbreite setzen
TTL	Titel setzen
SUBTTL	Setzt Untertitel
SPC	Leerzeilen ausgeben
PAGE	Seitenvorschub
LISTCHAR	Kontrollzeichen ausgeben
FORMAT	Definiert Listingformat

Labels

EQU	Wert Label setzen
EQU*	Register-Equate
REG	Definiert Registerliste
SET	Temporären Wert eines Labels setzen
RS	Reserviert Speicher
RSRESET	Reset RS-Zähler
RSSET	RS-Zähler setzen

Bedingtes Assemblieren

Assembliere, wenn

IFEQ	Null
IFNE	nicht null
IFD	Label definiert
IFND	Label nicht definiert
IFGT	größer als
IFGE	größer oder gleich
IFLT	kleiner als
IFLE	kleiner oder gleich
IFC	Strings gleich
IFNC	Strings nicht gleich
ELSEIF	Auf Assemblierung umschalten
IIFC	Immediate If
ENDC	Ende bedingtes Assemblieren

Makros

MACRO	Start der Definition
ENDM	Ende der Definition
MEXT	Erzwingt Ende Makroexpansion

Linkerdirektiven

IDNT	Setzt Modulname
SECTION	Sektionsname
XDEF	Definiert interne Labels als public
XREF	Definiert externe, zu importierende Labels
COMMENT	Wird z.Zt. ignoriert
ORG	Absolute Code-Erzeugung
OFFSET	Definiert Offsetliste

Reservierte Symbole

NARG	Zahl der Makroparameter
-G2	Interne Versionsnummer
-RS	RS-Zähler
-LK	Output-Dateityp

3.12 Der Linker BLink

Diese Version von BLink ist ein Public-Domain-Linker, der den Standard-Amiga-DOS-Linker ALink ersetzen soll. Das Programm besitzt viele Features. Die Kommandozeile muß die Form

```
blink [FROM] <infile1.o> [<infile2.o>...] [TO <outfile>]
```

besitzen, in der ein oder mehrere Inputdateien und eine Outputdatei spezifiziert werden. Normalerweise werden Sektionen separat gespeichert, sie können aber durch Eingabe von SMALLCODE oder SMALLDATA in der Kommandozeile zusammen gespeichert werden. Zum Beispiel wird durch

```
blink test.o
```

»test.o« in die Datei »test« gelinkt.

```
blink FROM start.o middle.o end.o TO total
```

linkt die drei spezifizierten Dateien in »total«. Der komplette Anweisungssatz für BLink ist im .doc-File auf Disk 2 enthalten, das entweder am Bildschirm gelesen oder durch folgende Methoden ausgedruckt werden kann:

3.12.1 Maschinen mit einem Laufwerk

Um die Datei am Bildschirm sehen zu können, geben Sie das Kommando

```
type devpac2:blink.doc
```

ein, oder Sie drucken sie mit

```
copy devpac2:blink.doc to prt:
```

3.12.2 Maschinen mit zwei Laufwerken

Legen Sie Disk 2 in das externe Laufwerk, und geben Sie entweder

```
type df1:blink.doc
```

ein oder

```
copy df1:blink.doc to prt:
```

Kapitel

4

MonAm, der symbolische Debugger

4.1 Einführung

Assemblerprogramme sind besonders empfindlich, weil schon der kleinste Fehler zum Absturz des Systems führen kann.

Um Ihnen zu helfen, diese Bugs zu finden und zu korrigieren, enthält das Devpac-Amiga-Paket MonAm. MonAm ist ein symbolischer Debugger und Disassembler, mit dem Sie Programme im Speicher untersuchen, Programme und einzelne Befehle ausführen und durch Programmfehler ausgelöste Exceptions des Prozessors abfangen können. Weil MonAm ein symbolischer Debugger ist, können Sie sich Ihr komplettes Programm mit allen Labels ansehen. Dadurch wird das Debuggen erheblich einfacher, da man nicht mit sechsstelligen Hex-Zahlen kämpfen muß.

Obwohl MonAm ein sogenannter Low-Level-Debugger ist, der zum Beispiel die 68000-Befehle oder Bytes im Speicher anzeigt, kann er auch auf Programme angewandt werden, die von einer beliebigen Hochsprache kompiliert wurden (und Maschinen-Code erzeugt haben). Wenn der Compiler die Option bietet, Symbole in das Binär-File zu schreiben, dann werden Sie auch Ihre Prozedur- und Funktionsnamen im Code sehen, natürlich nicht Original-Quelltext.

Weil MonAm seinen eigenen Bildschirmspeicher benutzt, wird das Display Ihres Programms nicht zerstört, wenn Sie es in Einzelschritten abarbeiten oder Break-points setzen. Das ist besonders bei grafisch orientierten Programmen wie Intuition-Anwendungen oder Spielen nützlich.

4.2 Exceptions des 68000

MonAm benutzt die 68000-Exceptions, um einen Absturz des zu testenden Programms abzufangen, sowie für die Einzelschritt-Abarbeitung Trace-Bits. Deshalb erscheint es sinnvoll, zu erläutern, was beim Amiga passiert, wenn Exceptions auftreten. Leider wurde von Commodore etwas Konfusion in die Sache gebracht, indem die Exceptions als Traps bezeichnet wurden, was aber in C und Assembler etwas

völlig anderes ist. Es gibt viele Arten von möglichen Exceptions. Einige davon sind (vom Programmierer) gewollt, andere nicht (Programmierfehler). Im Fall einer Exception geht der Prozessor in den Supervisor-Modus, rettet einiges auf dem Stack (SSP) und springt zur Exception-Routine. Wenn MonAm aktiv ist, richtet er einige Exception-Vektoren auf eigene Routinen, so daß er die Kontrolle übernehmen kann. Die verschiedenen Exceptions, ihr normales Ergebnis und das, was passiert, wenn MonAm aktiv ist, sind in der folgenden Tabelle dargestellt.

Exc.-Nr.	Bezeichnung	Normaler Effekt	Effekt unter MonAm
2	Bus-Fehler	Guru	Abgefangen
3	Adreß-Fehler	Guru	Abgefangen
4	Illegaler Befehl	Guru	Abgefangen
5	0-Division	Guru	Abgefangen
6	CHK-Befehl	Guru	Abgefangen
7	TRAPV	Guru	Abgefangen
8	Privilegverletzung	Guru	Abgefangen
9	Trace	Guru	Einzelschritt
10	Line-A-Emulator	Guru	Abgefangen
11	Line-F-Emulator	Guru	Abgefangen
32-47	Trap #0-15	Guru	Abgefangen

Die genauen Gründe für die oben genannte Exceptions (und wie man sie am besten behandelt) sind detailliert am Ende dieses Abschnitts beschrieben, kurz sei jedoch zusammengefaßt:

Exceptions 2 bis 8 werden durch einen Programmierfehler verursacht und von MonAm abgefangen. Exception 9 kann auf Umwegen durch einen Programmierfehler ausgelöst werden. Sie wird von MonAm für die Einzelschritt-Bearbeitung benutzt.

Der Rest (d.h. die Trap-Befehle) sind auf MonAm umgelenkt, können aber nachträglich undefiniert werden, wenn es das Programm erfordert. »Guru« in obiger Tabelle bedeutet, daß der Amiga versuchen wird, einen System-Alert-Requester auf den Schirm zu zeichnen, der die Exception-Nummer und den PC-Wert zeigt. Gelegentlich wird bei schweren Fehlern der ganze Schirm mit buntem Müll gefüllt, was zwar sehr beeindruckend aussieht, aber nicht sehr nützlich ist.

4.3 Starten von MonAm

MonAm wird gestartet durch die Eingabe des Kommandos

```
monam 
```

Dem kann wahlweise ein Programm-Name und eine Kommandozeile (für das Programm) folgen. Zum Beispiel:

```
monam c:genam examples/demo.s
```

startet MonAm, lädt GenAm und übergibt letzterem einen File-Namen. Nachdem MonAm geladen wurde, zeigt der Bildschirm vier Fenster, in denen MonAm alle Informationen darstellt. Jedes Fenster hat einen speziellen Zweck, doch vorerst ist das unterste Fenster das wichtigste. Hier erscheinen alle Tastatureingaben, während die anderen Fenster diverse Meldungen ausgeben. Direkt nach dem Laden erscheint die Aufforderung »Programmname oder Return eingeben:«. An dieser Stelle müssen Sie entscheiden, ob Sie ein Programm debuggen oder sich nur den Speicher ansehen wollen.

4.4 Programm debuggen

Wenn Sie ein bestimmtes Programm debuggen wollen, sollten Sie den Dateinamen einschließlich Laufwerksangabe und Extender eingeben. MonAm wird versuchen, die Datei zu laden. Schlägt das fehl, erscheint die Meldung

```
AmigaDOS error xx
```

Sie können dann einen anderen Dateinamen eingeben oder nur , wenn Sie Ihre Meinung geändert haben und kein Programm debuggen wollen. Wenn der Dateiname gültig ist, lädt ihn MonAm. Nach dem erfolgreichen Laden wird »Kommandozeile eingeben:« ausgegeben, und Sie können eine Kommandozeile eingeben, die wie üblich (d.h. in die Base Page) übergeben wird. Wenn Sie keine Kommandozeile wünschen, geben Sie nur ein. Nun kommt die Meldung

```
Exception: Breakpoint
```

und das Haupt-Display erscheint. Der Grund für den Breakpoint ist, daß MonAm einen Breakpoint auf den ersten Befehl des Programms setzt und dann dahin springt.

4.5 MonAm und Multitasking

Der Amiga ist multitaskingfähig, was für MonAm einige Beschränkungen mit sich bringt. Wurde ein Programm (eine Task) geladen, wird er in einen Wartezustand versetzt, das heißt in diesem Fall, er wartet auf ein MonAm-Kommando, das ihn

weiter ablaufen läßt. Der andere Zustand ist, daß die Task gleichzeitig mit MonAm läuft. Einige Kommandos funktionieren nur jeweils in einem dieser beiden Zustände. Zum Beispiel können Sie eine Task in Einzelschritten nur in Wartezustand abarbeiten. Läuft er, wenn Sie das Kommando (für single step) eingeben, kommt die Fehlermeldung »Unterbrochen!«.

4.6 Speicher ansehen

Wenn Sie sich den Speicher ansehen wollen, geben Sie nur **Return** ein, und Sie sehen das Haupt-Display und das Prompt-Kommando.

4.7 Symbolisch debuggen

Ein Hauptmerkmal von MonAm ist die Fähigkeit, die Originalsymbole des Programms beim Debuggen zu benutzen. MonAm unterstützt die Symbol-Hunks, wie sie vom Standard-AmigaDOS und von den meisten Amiga-Programmen erzeugt werden, die ausführbare Dateien wie Linker, Compiler und GenAm produzieren.

4.8 Dialog- und Alarmboxen

MonAm verwendet viele Dialog- und Alertboxen, die denen von Intuitions-Programmen entsprechen; es sind aber keine. Der Grund dafür ist einfach: MonAm muß vom Betriebssystem so unabhängig wie möglich sein; nur so ist es gewährleistet, daß MonAm auch unter den schwersten Bedingungen problemlos funktioniert. Eine MonAm-Dialogbox erkennt man daran, daß »ESC um abubrechen« über dem linken oberen Rand steht. In der Box steht der Mitteilungstext und (normalerweise) eine leere Zeile, in der sich der Cursor befindet. Man kann zu jedem Zeitpunkt mit **Esc** den Dialog abbrechen. Die Eingabe erfolgt über die Tastatur und die Cursortasten. Die **Backspace**- und **Del**-Taste können zum Editieren der eingetippten Zeile benutzt werden, **A-X** löscht die eingetippte Zeile bis zum Ende. Die Eingabe wird mit der **Return**-Taste beendet. Falls eine Zeile einen Fehler enthält, wird **Return** so lange ignoriert, bis die Zeile akzeptabel ist. Dialogboxen, in die mehr als eine Zeile eingegeben wird, lassen keine Zeilensprünge durch die Cursortasten zu. Eine MonAm-Alarmbox ist eine Box, die eine Meldung (meistens eine Fehlermeldung) anzeigt und auf **Return** oder **Esc** wartet.

4.9 Anfangs-Display

Sobald Sie im Editor die *Debuggen*-Option ausgewählt haben, erscheint eine Dialogbox, die Sie um die Eingabe eines Namens bittet. Wenn Sie ein Programm von Disk debuggen wollen, geben Sie den Dateinamen und `[Return]` ein. Sie werden dann nach einer Kommandozeile gefragt. Wenn Sie kein Programm debuggen wollen, aber zum Beispiel den Speicherplatz ermitteln wollen, drücken Sie `[Esc]` oder geben als Dateiname eine Anzahl Leerzeichen ein.

4.10 Das Haupt-Display

Das Haupt-Display von MonAm zeigt Register, einen Speicherbereich und Befehle. Das Anzeigenprinzip ist dem von alten Großrechnern nachempfunden: Damals befanden sich kleine flackernde Lämpchen an der Vorderseite des Rechners, damit man beobachten konnte, welche Register gerade benützt wurden. Dies sind Hardware-Anzeigen, MonAm zeigt Ihnen eine Software-Anzeige – der Code, in dem MonAm den Zustand Ihres Computers ausarbeitet und der dann auf dem Bildschirm angezeigt wird.

Die MonAm-Anzeige bei Programmstart besteht aus drei Fenstern: Das obere Fenster (Fenster 1) zeigt die Daten und Adreßregister an, und den Speicher, auf den diese Register zeigen. Das nächste Fenster (Fenster 2) ist das Disassemblierungsfenster, in dem mehrere Zeilen disassemblierten Codes zu sehen sind. Die Anfangsadresse des Fensters ist normalerweise der PC; A zeigt die momentane Position des PC. Fenster 3 zeigt einen Teil des Speichers in Hex und ASCII an. Der Bereich am unteren Rand des Bildschirms wird für die Anzeige von Meldungen verwendet.

Eine der mächtigsten Funktionen von MonAm ist seine Flexibilität in der Fensteranzeige – bis zu zwei zusätzliche Fenster können erzeugt, die Fontgröße kann verändert, und Fenster können an bestimmte Register gebunden werden. Diese Features werden später erläutert.

4.10.1 Benutzung der Fenster

MonAm hat immer ein aktuelles Fenster. Dieses Fenster erkennt man daran, daß der Fenstertitel weiß auf schwarz, statt umgekehrt, erscheint. Die `[Tab]`-Taste kann dazu verwendet werden, durch die einzelnen Fenster zu gehen, man kann aber auch die `[A]`-Taste zusammen mit der Fensternummer drücken, um ein bestimmtes Fenster zum aktuellen Fenster zu machen. Das kleine Fenster unten kann nie das aktuelle Fenster werden, da es nur zur Anzeige von Mitteilungen gedacht ist.

Hinweis: Falls Ihre Eingaben von MonAm anscheinend ignoriert werden, bedeutet das, daß ein anderes Fenster aktiv ist. Um das zu ändern, klicken Sie mit der Maus irgendwo in der MonAm-Anzeige.

4.11 Eingabe von Befehlen

Der MonAm-Befehlssatz basiert auf einzelnen Tasten, um ein schnelles Arbeiten zu ermöglichen. Dies ist zu Anfang natürlich etwas gewöhnungsbedürftig, macht sich aber schon nach geringer Zeit bezahlt. Allgemein gilt, daß die **[A]**-Taste die Fenstertaste ist. Alle Befehle, die die **[A]**-Taste beinhalten, haben Auswirkungen auf Fenster.

Befehle können groß oder klein geschrieben eingegeben werden. Alle Befehle, die unter Umständen katastrophale Auswirkungen haben können, erfordern die **[Ctrl]**-Taste. Befehle werden sofort ausgeführt, es braucht nicht nach jedem Befehl **[Return]** zusätzlich eingegeben zu werden. Ungültige Befehle werden ignoriert. Die Fenster, deren Daten sich verändert haben könnten, werden auch nach jedem Befehl auf den neuesten Stand gebracht.

MonAm ist ein sehr funktionsstarkes und manchmal komplex erscheinendes Programm. Wir sehen natürlich ein, daß es unwahrscheinlich ist, daß jeder Benutzer jeden einzelnen Befehl auf Anhieb gebrauchen wird. Deshalb ist dieses Kapitel in zwei weitere Teile gegliedert: einmal eine Einleitung in die Benutzung MonAms und darauf folgend ein vollständiger Referenzteil. Es ist ohne weiteres möglich, daß Anfänger effektiv den Debugger benutzen, auch wenn sie nur die Bedienungsanleitung gelesen haben. Lassen Sie sich aber nicht vom Referenzteil abschrecken: Er enthält sehr nützliche und wichtige Informationen.

4.12 Überblick über MonAm

Zuerst müssen Sie das Programm laden, das debuggt werden soll. Wenn Sie ein Programm in den Speicher assembliert haben, benutzen Sie die *Debuggen*-Option im Editor, ansonsten müssen Sie das Programm von Disk laden. Nachdem Sie den Ladebefehl eingegeben haben, werden Sie nach einem Dateinamen gefragt. Wenn daraufhin ein Fehler gemeldet wird oder kein Dateiname spezifiziert wurde, können Sie durch **[Ctrl]-[L]** weitere Namen aufrufen. Vom Debugger werden auch Programmsymbole benutzt. Ein Programm enthält Symbole, wenn Sie im Assembler oder Linker die *Debuggen*-Option verwenden.

Eines der wichtigsten Kommandos in MonAm ist das Single-Steppen, das durch **[Ctrl]-[Z]** (oder **[Ctrl]-[Y]**) aufgerufen werden kann. Dadurch wird die Anweisung am PC ausgeführt, der im Registerfenster und normalerweise auch im Disassemblier-

rungsfenster gezeigt wird. Nachdem der Befehl ausgeführt wurde, zeigt der Debugger nochmals alle Werte der Register an, so daß Sie Schritt für Schritt beobachten können, wie Ihr Programm ausgeführt wird. Das Single-Steppen ist die beste Methode, sich durch bestimmte Abschnitte des Source-Codes zu bewegen und gleichzeitig eine Überprüfung vorzunehmen. Allerdings ist diese Methode die langsamste – bis Sie zu der Stelle kommen, die Ihnen vielleicht verdächtig erscheint, benötigen Sie viel Zeit und Geduld, da das gesamte Programm single-gestept werden muß. Jedoch gibt es auch für dieses Problem eine Lösung.

Ein Breakpoint ist ein spezielles Wort, das in das Programm eingebunden wird, um dessen Ablauf zu beenden und MonAm aufzurufen. Es gibt eine ganze Menge von verschiedenen Breakpoints, wir werden uns aber in unseren Ausführungen auf den einfachsten Typ beschränken. Ein Breakpoint kann durch die Eingabe von **[A]-[B]** mit einer danach folgenden Angabe seiner Adresse gesetzt werden. Sie können in MonAm Adressen in Hex (der Grundeinstellung) als Symbol oder als komplexen Ausdruck eingeben. Beispiele für korrekte Adressen sind »1A2B0, prog_start, 10+mydata«. Bei der Eingabe einer falschen Adresse flackert der Bildschirm, worauf Sie eine Korrektur vornehmen können.

Nachdem Sie einen Breakpoint gesetzt haben, müssen Sie Ihr Programm ablaufen lassen. Dies erreichen Sie mit **[Ctrl]-[R]**. Ihr Programm läuft dann mit Beginn am PC ab unter Verwendung der angezeigten Register. MonAm wird wieder aufgerufen, wenn ein Breakpoint angetroffen wurde oder eine Exception geschieht.

MonAm benutzt seine eigene Bildschirmanzeige, die unabhängig ist von Ihren Programmen. Wenn Sie die Gadgets am oberen Rand des Bildschirms benutzen, sehen Sie die aktuelle Programmanzeige. Damit können Sie Programme debuggen, ohne deren Output zu zerstören.

Die Fenster von MonAm können mittels Eingabe von **[A]-[Z]** auf volle Bildschirmgröße gezoomt werden. Um in den Hauptbildschirm zurückzukehren, drücken Sie **[A]-[Z]** oder **[Esc]**. Durch **[Esc]** kommen Sie aus fast allen Situationen wieder heraus, in die Sie durch Zufall hineingeraten sind. Das Zoom-Kommando arbeitet wie alle **[A]**-Kommandos auf dem aktuellen Bildschirm, den Sie durch die **[Tab]**-Taste verändern können. Durch Eingabe von **[A]-[P]** können Sie den Inhalt des aktuellen Fensters zum Drucker senden.

Zum Ändern der Adresse, von der aus ein Fenster seine Daten anzeigt, geben Sie **[A]-[A]** ein und ändern dann die Adresse um. Beachten Sie, daß das Disassemblierungsfenster am PC festgemacht ist und deshalb nach dem Single-Steppen immer vom PC aus angezeigt wird.

Um MonAm zu verlassen, geben Sie `Ctrl-C` ein. MonAm kehrt daraufhin direkt zum CLI zurück. Falls Sie versuchen sollten, das Programm zu verlassen, wenn gleichzeitig noch ein Debug ausgeführt wird, werden Sie gewarnt. Wenn MonAm während eines Debugs aussteigt und danach Exceptions auftreten, stürzt die Maschine unweigerlich ab. Benutzen Sie vorher also lieber `Ctrl-Q`, um die Debug-Anweisung zu beenden. Wenn Sie die *Debuggen*-Option vom Editor aus benutzen, werden durch `Ctrl-C` sowohl MonAm als auch Ihr Programm beendet.

Wir hoffen, daß Ihnen dieser Abschnitt einen Überblick über die wichtigsten Eigenschaften von MonAm gewährt hat, so daß Sie den folgenden Kapiteln, in denen Sie in den Prozeß der Erstellung und des Debuggens von Assemblerprogrammen eingeweiht werden, ohne Schwierigkeiten folgen können.

4.13 MonAm – Referenzteil

4.13.1 Numerische Ausdrücke

MonAm kann numerische Ausdrücke auswerten, ähnlich wie in GenAm. Der Hauptunterschied zu GenAm liegt darin, daß MonAms normale Zahlenbasis Hex-Code ist, dezimalen Zahlen muß ein `»$«`-Zeichen (kein `»#«` wie in Version 1!) vorangesetzt werden. Es gibt keine Ausdruckstypen (absolut oder relativ), das `»*«`-Zeichen wird nur für die Multiplikation verwendet. Es gibt kein Gleichheitszeichen `»<>«`.

Es können Symbole verwendet werden, die normalerweise case-sensitiv sind, d.h., daß ein Unterschied zwischen kleinen und großen Buchstaben gemacht wird, und die eine Signifikanz bis 32 Zeichen haben (je nach Symbolformat). Diese Standardeinstellungen können mit *Voreinstellungen* verändert werden. Register werden mit ihren Namen bezeichnet, z.B. A3 oder D7. Dies sind aber keine Hex-Zahlen. Um die Hex-Zahl A0 anzugeben, muß entweder ein `»$«`-Zeichen der Zahl vorangesetzt werden oder die Zahl Null. A7 bezeichnet immer den User-Stack-Pointer.

Es gibt reservierte Symbole, die nicht case-sensitiv sind: CODE, SP, SR und SSP. CODE bezeichnet das erste Hunk im Programm, nämlich HUNK1. Das zweite Hunk ist HUNK2 usw. Es gibt 10 Speicher, M0 bis M9, ähnlich wie bei Taschenrechnern. Diese Speicher können, wie Register, Werte zugewiesen bekommen. Die Speicher 2 bis 5 enthalten immer die Startadresse der Fenster 2 bis 5; wenn Sie einem dieser Speicher einen neuen Wert zuweisen, verändert sich auch die Anfangsadresse dieses Fensters.

Vorsicht: Weisen Sie nie M4 einen neuen Wert zu, wenn Fenster 4 als Source-Fenster benutzt wird.

Ausdrücke können auch Indirektionen beinhalten: die Zeichen »{« und »}« werden dafür benutzt. Indirektion kann auf einer Byte-, Wort- oder Langwortbasis angegeben werden, wenn »}« von einem Punkt und der gewünschten Größe gefolgt wird; der Normalwert ist das Langwort. Falls der Pointer ungültig ist, entweder weil der Speicher unlesbar oder ungerade ist (wenn Wort- oder Langwortgröße angegeben ist), ist der gesamte Ausdruck ungültig. Zum Beispiel gibt der Ausdruck

```
{data_start+10}.w
```

die Wortinhalte an der Adresse »data_start+10« an, vorausgesetzt, »data_start« ist eine gerade Adresse. Indirektionen werden ähnlich verschachtelt.

4.13.2 Fenstertypen

Es gibt vier verschiedene Fenstertypen. Die erlaubten Fenstertypen sind wie folgt:

Fensternummer	Erlaubte Typen
1	Register-Fenster
2	Disassemblierungs-Fenster
3	Speicher-Fenster
4	Disassemblierung, Speicher oder Source-Code
5	Speicher

Register-Fenster-Anzeige

In einem Register-Fenster werden die Inhalte der Datenregister in Hex angezeigt, zusätzlich der ASCII-Wert des untersten Bytes und die Anzeige (in Hex) der ersten 8 Byte, auf die der Registerinhalt zeigt. Die Adreßregister werden auch in Hex angezeigt, zusammen mit den ersten 12 Byte, auf die das Register zeigt. Wie in allen Hex-Anzeigen in MonAm werden Werte in nicht lesbaren Speicherstellen mit »**« ersetzt.

Das Statusregister wird sowohl in Hex wie auch als Flags dargestellt. Zusätzlich wird entweder »U« oder »S« angezeigt, je nachdem, ob der Rechner sich im Supervisor- oder Usermodus befindet. »A7« ist der Supervisor-Stack-Pointer und wird wie ein normales Adreßregister angezeigt. Der Wert des PC wird zusammen mit der Instruktion, die am PC ist, in der untersten Zeile des Register-Fensters angezeigt. Wenn die Instruktion ein oder zwei effektive Adressen hat, werden diese zusammen mit dem Speicher, der auf diese Adresse zeigt, angezeigt. Zum Beispiel bedeutet die Zeile

TST.W \$12A(A3) ;00001FAE 0F01

daß der Wert von »A3« mit »\$12A« addiert die Adresse »\$1FAE« ergibt. An dieser Adresse ist der Wert »\$0F01«. Ein komplexeres Beispiel ist

MOVE.W \$12A(A3),-(SP) ;00001FAE 0F01 0002AC08 FFFF

Der Ursprungs-Adreßmodus ist der gleiche wie im ersten Beispiel, die Zieladresse ist dagegen »\$2AC08«, die z. Zt. »\$FFFF« enthält. Beachten Sie, daß die Anzeige immer so groß ist wie gerade nötig. MOVEM-Daten werden als Quad-Worte (64 Bit) angezeigt. Wenn Prädecrement im Adressierungsmodus verwendet wird, wird dies bei der Adreßberechnung mit in Betracht gezogen.

In niedriger Auflösung werden keine Hex-Daten für die Datenregister und nur vier Byte als Daten bei Adreßregistern angezeigt. Es kann auch in niedriger Auflösung vorkommen, daß die Bildschirmbreite nicht dafür ausreicht, um eine komplexe Zeile (wie das zweite Beispiel) anzuzeigen.

Disassemblierungs-Fenster

In einem Disassemblierungs-Fenster wird Speicherinhalt in disassemblierter Form angezeigt. Ganz links auf einer Zeile wird die Adresse angezeigt, dann gegebenenfalls das Symbol an dieser Adresse, daraufhin der Befehl an dieser Adresse. Der aktuelle Wert des PC wird mit »>>« angezeigt. Falls an dieser Adresse ein Breakpoint gesetzt ist, wird dessen Typ in eckigen Klammern ([]) nach dem Befehl angezeigt. Bei Stop-Breakpoints wird angezeigt, wie oft dieser Befehl noch ausgeführt werden muß, bis der Breakpoint aktiv wird und das Programm stoppt. Bei einem Breakpoint mit einer Bedingung enthalten die eckigen Klammern ein Fragezeichen, gefolgt von der Bedingung. Bei Zähler-Breakpoints enthalten die Klammern ein Gleichheitszeichen und den Zählerwert. Bei permanenten Breakpoints ist ein »*« in den eckigen Klammern.

Das Format der disassemblierten Instruktionen ist im Motorola-Standard, wie GenAm ihn akzeptiert. Alle Textdaten sind groß geschrieben, bis auf die Symbole, die kleine Buchstaben enthalten. Alle numerischen Daten sind in Hex, bis auf die TRAP-Nummern. Vorangehende Nullen und der Hex-Bezeichner »\$« werden bei Zahlen kleiner als 10 nicht angezeigt. Wenn erforderlich, werden numerische Daten mit Vorzeichen angezeigt. Die einzige Abweichung vom Motorola-Standard ist die Darstellung der Registerliste des MOVEM-Befehls:

MOVEM.L d0-d3/a0-a2,-(sp)

wird, um Platz zu sparen, als

MOVEM.L d0-3/a0-2,-(sp)

disassembliert.

In niedriger Auflösung ersetzen Symbole die Adresse und werden nur bis zu einer maximalen Länge von acht Zeichen angezeigt. Gewisse Library-Aufrufe werden symbolisch angezeigt, wenn zusammen mit dem Programm keine Symbolinformation geladen wurde. Der Disassembler ist intelligent und erkennt einen MOVE in das Register A6, das von einem JSR gefolgt wird, das A6 gebraucht. Nach dem Erkennen wird der Aufruf namentlich angezeigt, wie z.B. in

```
MOVE.L 4,A6
```

```
JSR _LV00openLibrary(A6)
```

Dies geschieht in den Exec-, Grafik- und Intuition-Libraries, die die spezielle Datei »libs:libfile.monam« benutzen. Wenn eine solche Datei während der Initialisierung von MonAm nicht gefunden wird, wird eine Disassemblierung nicht ausgeführt.

Speicher-Fenster

In einem Speicher-Fenster wird Speicher zeilenweise im Format Hex-Adresse, Hex-Daten und ASCII-Daten angezeigt. Unlesbarer Speicher wird als »**« dargestellt. Die Anzahl der angezeigten Bytes pro Zeile hängt von der Breite des Fensters ab und kann maximal 16 Byte betragen.

Source-Fenster

Sie können im Source-Fenster ASCII-Dateien, wie z.B. Ihren Programm-Quelltext, ähnlich wie in einem Text-Editor darstellen. Der normale Tab-Abstand ist 8, kann aber mit dem Editieren-Fenster-Befehl auf 4 umgeschaltet werden.

4.13.3 Fenster-Befehle

Die -Taste ist die sogenannte Fenster-Taste, d.h., alle Fenster-Befehle beinhalten die -Taste. Jeder dieser Befehle bezieht sich auf das aktuelle Fenster, also auf das Fenster, dessen Titel invers dargestellt ist. Das aktuelle Fenster kann gewählt werden, indem man entweder die -Taste so oft drückt, bis das gewünschte Fenster zum aktuellen Fenster wird, oder man drückt  zusammen mit der Nummer des gewünschten Fensters. Die meisten Fenster-Befehle funktionieren in allen Fenstern, unabhängig davon, ob sie sich im Zoom-Modus befinden oder nicht. Wenn ein Befehl keinen Sinn ergibt, wird er ignoriert.

- Adresse setzen

Dieser Befehl setzt die Anfangsadresse eines Fensters.

[A]-[B] Breakpoint setzen

Mit diesem Befehl werden Breakpoints gesetzt; Breakpoints werden später genauer beschrieben.

[A]-[E] Editieren

In einem Speicher-Fenster kann man mit diesem Befehl im Speicher in Hex und ASCII editieren. Hex-Zahlen können mit den Tasten 1–9 und A–F eingegeben werden. Die Cursortasten können dazu benutzt werden, den Cursor im Fenster zu bewegen. Mit der [Tab]-Taste können Sie zwischen dem Hex- und dem ASCII-Editiermodus hin- und herschalten. Der ASCII-Modus schreibt den ASCII-Wert des Tastendruckes in den Speicher. Um den Editiermodus zu verlassen, drücken Sie die [Esc]-Taste.

Im Register-Fenster bewirkt [A]-[E] das gleiche wie [A]-[R], nämlich das Setzen von Registern. Im Source-Fenster kann mit [A]-[E] der Tab-Abstand zwischen 4 und 8 gewählt werden.

[A]-[L] Fenster binden

Mit diesem Befehl kann ein Fenster an ein Register oder an ein anderes Fenster gebunden werden. Nach einer Exception werden die Startadressen aller Fenster neu berechnet und ggf. entsprechend verändert. Um eine Bindung zu löschen, geben Sie in die Dialogbox einen Leerstring ein. Fenster 2 ist standardmäßig an den PC gebunden. Um ein Fenster an ein anderes zu binden, müssen Sie den entsprechenden Speicher angeben, z.B. M2 für Fenster 2.

[A]-[O] Auswerten

Eine Dialogbox erscheint, in der Sie einen Ausdruck eingeben. Dieser wird ausgewertet und in Hex, Dezimal und ggf. als Symbol ausgegeben.

[A]-[P] Drucker-Dump

Der Inhalt des aktuellen Fensters wird auf dem Drucker ausgegeben. Dieser Befehl kann mit [Esc] abgebrochen werden.

[A]-[R] Register setzen

Mit diesem Befehl kann jedes Register gesetzt werden. Das Register muß angegeben werden, gefolgt von einem Gleichheitszeichen und dem Ausdruck des neuen Registerwerts. Zum Beispiel setzt

A3=A2+4

das Register »A3« auf den Wert von Register »A2« mit 4 addiert. Sie können z.B. im Zoom-Modus diesen Befehl dazu verwenden, einem anderen Fenster eine neue

Startadresse zu geben. Wenn Sie den Zoom-Modus verlassen, zeigt das Fenster auf die angegebene Adresse.

[A]-[S] Fenster spalten

Dieser Befehl spaltet Fenster 2 in die Fenster 2 und 4 oder das Fenster 3 in die Fenster 3 und 5. Jedes neue Fenster ist vom Ursprungsfenster unabhängig. Wenn Sie [A]-[S] ein zweites Mal drücken, werden die zwei Fenster wieder zusammengefügt. Dieser Befehl wird in niedriger Auflösung ignoriert.

[A]-[T] Typ verändern

Dieser Befehl hat nur Wirkung in Fenster 4, das durch die Spaltung von Fenster 2 entsteht, oder wenn Sie eine ASCII-Datei laden. Der Fenstertyp wird zwischen Disassemblierung, Speicheranzeige oder Source umgeschaltet.

[A]-[Z] Zoom

Das aktuelle Fenster wird mit diesem Befehl auf die volle Bildschirmgröße gebracht. Mit [Esc] oder einem erneuten [A]-[Z] wird das Fenster wieder auf Normalgröße gebracht. [A]-Befehle sind im Zoom-Modus weiterhin benutzbar.

Cursortasten

Die Cursortasten werden in Verbindung mit dem aktuellen Fenster benutzt. Ihre Wirkung hängt vom Fenstertyp ab. In einem Speicher-Fenster verändern alle vier Cursortasten die Startadresse des Fensters. Mit [Shift]-[↑] und [↓] können Sie seitenweise nach oben und nach unten blättern. In einem Disassemblierungs-Fenster verändern [↑] und [↓] die Startadresse um jeweils eine Instruktion. [→] und [←] ändern die Adresse auf Wortbasis. Im Source-Fenster verändern [↑] und [↓] die Anzeigen zeilenweise, [Shift]-[↑] und [↓] seitenweise.

4.13.4 Bildschirmumschaltung

MonAm benutzt seinen eigenen Bildschirmspeicher und seinen eigenen Bildschirmtreiber, um jeden Konflikt mit dem zu debuggenden Programm zu vermeiden. Um ein Flackern des Bildschirms beim Single-Steppen zu verhindern, wird von der Bildschirmanzeige auf die des Programms erst nach 20 ms umgeschaltet. Es ist auch möglich, den Debugger in einer anderen Auflösung als der für das zu debuggende Programm zu betreiben; dies geht aber nur mit einem Farbmonitor. Um den Programm-Output betrachten zu können, benutzen Sie die Fenster-Gadgets Ihres Amiga. Wenn das Programm auf dem Workbench-Bildschirm abläuft, können Sie es durch die linke [A]-[N] anzeigen lassen, mit linker [A]-[M] kehren Sie in das MonAm-Fenster zurück.

4.13.5 Breakpoints

Breakpoints erlauben Ihnen, das Programm an von Ihnen bestimmten Punkten zu unterbrechen. MonAm erlaubt maximal acht gleichzeitig gesetzte Breakpoints, von denen jeder einer der fünf Breakpoint-Typen sein kann. Wenn ein Breakpoint erreicht wird, trifft MonAm die Entscheidung, ob das Programm angehalten wird oder weiterläuft. Diese Entscheidung hängt von den Typen des Breakpoints ab.

Einfache Breakpoints

MonAm hält das Programm bei Erreichen der Breakpoints an und löscht dann den Breakpoint.

Stop-Breakpoints

Diese Art unterbricht das Programm an der Stelle, an der ein bestimmter Befehl an der Adresse des Breakpoints zum n-ten Mal ausgeführt wurde. Ein einfacher Breakpoint ist im Grunde genommen ein Stopp-Breakpoint mit dem Wert 1.

Zähler-Breakpoints

Diese Art ist nur ein Zähler. Jedesmal, wenn der Befehl an der Adresse des Breakpoints ausgeführt wird, erhöht sich der Zähler um 1, und das Programm läuft weiter.

Permanente Breakpoints

Dieser Typ ist den einfachen Breakpoints ähnlich, bis auf den Unterschied, daß sie nie gelöscht werden. Jedesmal, wenn dieser Breakpoint erreicht wird, übernimmt MonAm die Kontrolle.

Bedingte Breakpoints

Diese Art von Breakpoint erlaubt es Ihnen, Ihr Programm nur dann anzuhalten, wenn eine bestimmte Bedingung erfüllt ist. Jeder bedingte Breakpoint hat einen Ausdruck; dieser wird jedesmal, wenn der Breakpoint erreicht wird, ausgewertet. Wenn der Ausdruck wahr (d.h. ungleich Null) ist, wird der Programmablauf unterbrochen.

- Breakpoint setzen

Dies ist ein Fenster-Befehl; er erlaubt das Setzen oder Löschen von Breakpoints zu jeder Zeit. Die eingegebene Zeile muß je nach Breakpoint-Typ eines der folgenden Formate haben:

<Adresse>

setzt einen einfachen Breakpoint.

<Adresse>, <Ausdruck>

setzt einen Stopp-Breakpoint an der angegebenen Adresse, der das Programm stoppt, nachdem <Ausdruck> eine bestimmte Anzahl Mal ausgeführt wurde.

<Adresse>, =

setzt einen Zähler-Breakpoint. Der Anfangswert des Zählers ist Null.

<Adresse>, *

setzt einen permanenten Breakpoint.

<Adresse>, ?<Ausdruck>

setzt einen bedingten Breakpoint, der von <Ausdruck> abhängig ist.

<Adresse>, -

löscht jede Art von Breakpoint an der angegebenen Adresse.

Breakpoints können nicht an ungeraden und unlesbaren Adressen und im ROM gesetzt werden. ROM-Breakpoints können emuliert werden, indem man den Run-Until-Befehl benutzt. Jedesmal, wenn ein Breakpoint erreicht wird, egal, ob das Programm angehalten wird oder nicht, wird der Prozessorstatus im History-Buffer, der später beschrieben wird, behalten.

Help Breakpoint- und Segmentanzeige

Dieser Befehl zeigt die Adressen der Text-, Daten- und BSS-Segmente und deren Längen zusammen mit sämtlichen gesetzten Breakpoints an. [A]-Befehle können hier auch benutzt werden.

[Ctrl]-[B] Breakpoint setzen

Dieser Befehl ist hauptsächlich aus Kompatibilitätsgründen zu MonAm1 noch vorhanden. Hiermit wird ein einfacher Breakpoint an der Startadresse des aktuellen Fensters gesetzt. Falls schon ein Breakpoint an dieser Adresse vorhanden ist, egal welcher Art, wird er gelöscht.

[U] Go Until

Dieser Befehl fragt in einer Dialogbox nach einer Adresse, an der dann ein einfacher Breakpoint gesetzt wird. Das Programm läuft danach sofort weiter ab.

[Ctrl]-[K] Kill Breakpoints

Alle gesetzten Breakpoints werden gelöscht.

[Ctrl]-[A] Breakpoint setzen und ausführen

Mit diesem Befehl wird ein einfacher Breakpoint nach dem Befehl am PC gesetzt, das Programm läuft weiter. Dieser Befehl ist hauptsächlich für DBF-Schleifen gedacht, die man nicht immer wieder durcharbeiten will, weil man das Ergebnis gleich sehen möchte.

[Ctrl]-[X] Abbruch der Programmausführung

Mit diesem Kommando wird der Ablauf Ihres Programms abgebrochen. Da das Trace-Bit gesetzt wird, erhalten Sie eine Trace-Exception. Beachten Sie bitte, daß Sie das Programm nicht während bestimmter AmigaDOS-Routinen unterbrechen.

Hinweis: Die im obigen Befehl beanspruchten Speicherfelder sind nicht unbedingt in jeder Version des Amiga-Betriebssystems die gleichen. Sie sollten deshalb vorher eine Kompatibilitätsprüfung vornehmen.

4.13.6 History

MonAm hat einen sogenannten History-Speicher, in dem der Prozessorstatus nach jeder Exception behalten wird. Die häufigste Ursache eines Eintrags in den History-Speicher ist das Single-Steppen. Aber auch bei jedem Breakpoint und bei bestimmten Arten des *Ausführen*-Befehls wird ein neuer Eintrag in den Speicher gemacht. Der Speicher enthält Platz für fünf Einträge. Wenn er voll ist, wird jeweils der älteste Eintrag gelöscht, damit ein neuer aufgenommen werden kann.

[H] History-Speicher-Anzeige

Ein Fenster wird geöffnet, in dem die maximal fünf Einträge im History-Speicher angezeigt werden. Sowohl alle Registerwerte als auch der nächste Befehl, der ausgeführt wird, werden angezeigt. Wenn auf einen Befehl im History-Speicher ein Breakpoint gesetzt ist, zeigt der Wert in den eckigen Klammern den Wert des Breakpoints zur Zeit der Anzeige und nicht zum Zeitpunkt des Eintrags in den Speicher.

4.13.7 MonAm verlassen

[Ctrl]-[C] Abbruch

Dieser Befehl verursacht einen Programmende-Trap für den aktuellen GEM-DOS-Task. Wenn ein Programm mit MonAm debuggt wird, wird zuerst das Programm mit [Ctrl]-[C] abgebrochen, und die Meldung »Programmende« erscheint am unteren Bildschirmrand im Mitteilungsfenster; durch nochmaliges [Ctrl]-[C] wird MonAm verlassen. Sie können aber auch ein anderes Programm laden, ohne

MonAm verlassen zu müssen. Wenn MonAm vom Editor aus aufgerufen wurde, kehren Sie sofort nach Beendigung des Programms in den Editor zurück, ohne daß Sie **Ctrl**-**C** für MonAm drücken müssen.

Vorsicht: Es kann durchaus zum Absturz kommen, wenn Sie frühzeitig ein GEM-Programm abbrechen, ohne daß z.B. die Virtual Workstation geschlossen wurde.

Ctrl-**Q** Programm verlassen

Mit diesem Kommando können Sie das zu debuggende Programm verlassen. Allerdings sollten Sie bedenken, daß ein eventuell angegebener Speicherplatz nicht deallokiert wird, wenn Sie auf diese Weise aussteigen. Das oben angegebene Kommando beansprucht Speicherfelder, die nicht unbedingt bei jeder AmigaDOS-Version identisch sind. Sie sollten also zuerst vorsichtshalber einen Probelauf machen, bevor Sie sich an eine ernsthafte Anwendung wagen.

4.13.8 Laden und Abspeichern

Ctrl-**L** Ausführbares Programm laden

Mit diesem Befehl wird ein Programm zum Debuggen geladen. Sie werden nach dem Programmnamen (wenn nicht anders angegeben, wird .PRG als Extension angenommen) und dann nach der Kommandozeile gefragt, die dem Programm übergeben werden soll. Wenn MonAm schon ein Programm geladen hat, muß dieses Programm erst beendet werden, bevor ein neues geladen werden kann. Die Datei, die geladen werden soll, muß ein ausführbares Programm sein. Wenn Sie andere Dateitypen editieren wollen, müssen Sie sie als Binärdatei laden.

B Binärdatei laden

Es wird nach einem Dateinamen und nach einer (nicht notwendig anzugebenden) Ladeadresse gefragt. Diese Datei wird in den Speicher geladen. Wenn keine Ladeadresse angegeben wird, wird die Datei dort geladen, wo GEM-DOS genügend Speicher freigibt. M0 enthält die Startadresse, M1 die Adresse des Dateiendes im Speicher.

S Binärdatei sichern

Dieser Befehl fragt nach einem Dateinamen, unter dem die Datei gesichert werden soll, nach einer Startadresse und einer Schlußadresse. Um eine zuvor geladene Binärdatei zu sichern, können Sie als Start- und Schlußadresse M0 und M1 angeben, vorausgesetzt, die Werte haben sich nicht verändert.

[A] ASCII-Datei laden

Mit diesem Befehl können Sie eine ASCII-Datei laden, normalerweise eine Quelltextdatei, um sie in MonAm anzusehen. Fenster 4 wird geöffnet, falls es noch nicht vorhanden ist, und als Source-Fenster eingestellt.

4.13.9 Programme ausführen

[Ctrl]-[R] Zurück zum Programm / laufenlassen

Hiermit wird das Programm mit voller Geschwindigkeit laufen gelassen; es ist die übliche Methode dafür, das Programm nach einem Breakpoint weiterlaufen zu lassen.

[Ctrl]-[Y] /**[Ctrl]-[Z] Single-Step**

Hiermit wird der Maschinensprachebefehl am PC ausgeführt. Ein Trap, Line-A- oder Line-F-Opcode wird als einzelner Befehl angesehen und als solcher ausgeführt. Dies kann durch Voreinstellungen verändert werden.

[Ctrl]-[T] Befehl interpretieren

Dieser Befehl interpretiert den Opcode am PC. Er ist dem **[Ctrl]-[Y] / [Ctrl]-[Z]** ähnlich, verfolgt aber keine JSR-, BSR-, Trap-, Line-A- oder Line-F-Befehle. Er erspart Ihnen das Durchlaufen solcher Routinen und kann mit Befehlen im ROM oder im RAM benutzt werden.

[Ctrl]-[S] Befehl überspringen

[Ctrl]-[S] inkrementiert das PC-Register um die Größe der aktuellen Instruktion und bewirkt so, daß sie übergangen wird. Sie sollten diesen Befehl anstatt **[Ctrl]-[Z]** benutzen, wenn Sie von Anfang an wissen, daß dieser Befehl ungewollte Aktionen auslöst.

[R] Run

Dies ist ein allgemeiner Befehl, der auf verschiedene Arten ausgeführt werden kann.

Run G Go

Dieser Befehl ist mit **[Ctrl]-[R]** identisch und läßt das Programm mit voller Geschwindigkeit laufen.

Run I Instruktion

Dieser Befehl ist dem »Run S« ähnlich, nimmt aber einen numerischen Parameter: die Anzahl der Instruktionen, die ausgeführt werden, bevor MonAm wieder eingreift.

4.13.10 Speicher durchsuchen

G Speicher durchsuchen

Nach diesem Befehl erscheint »Suchen nach B/W/L/T/I«, wobei die Kürzel Byte, Wort, Langwort, Text oder Instruktion bedeuten.

Wenn Sie B, W oder L wählen, werden Sie nach den Zahlensequenzen gefragt, nach denen Sie suchen wollen. MonAm findet auch Zahlen an ungeraden Adressen. Wenn Sie T wählen, geben Sie den Text ein, nach dem gesucht werden soll, er wird dann case-sensitiv behandelt. Geben Sie I ein, so können Sie nach einem ganzen oder nur nach einem Teil eines Maschinensprachebefehls suchen. Wenn Sie z.B. nach »\$14(A« suchen, werden Sie eine Instruktion wie »MOVE.L D2,\$14(A0« finden. Anders wie bei MonAm1 wird zwischen Groß- und Kleinschreibung unterschieden. Sie sollten auch das Ausgabeformat des Disassemblers beachten: So sollten Sie z.B. für Zahlen Hex-Code verwenden, sich auf A7 anstatt SP beziehen usw.

Wenn Sie die Parameter eingegeben haben, wird die Kontrolle an den N-Befehl übergeben.

N Nächstes Vorkommnis suchen

Dieser Befehl wird nach dem G-Befehl benutzt; er sucht nach dem nächsten Vorkommnis der angegebenen Suchparameter. Wenn Sie nach Byte-, Wort- oder Langwort-Daten suchen, werden Sie immer ein Vorkommnis, nämlich im MonAm-Speicher, finden. Wenn Sie nach Text suchen, kann es auch sein, daß der Text sich noch im Tastaturpuffer befindet. Mit diesen Suchparametern wird nach 64 Kbyte durchsuchtem Speicher die Esc-Taste abgefragt. Bei der Suche nach Maschinensprachebefehlen wird immer nach 2 Byte auf die Esc-Taste hin geprüft.

Gesucht wird im Bereich von 0 bis zum Ende des Speichers, dann von \$FA0000 bis \$FEFFFF, dem Modul- und System-ROM Bereich, und dann wieder von 0 an. Die Suche beginnt kurz nach der Anfangsadresse des aktuellen Fensters (wenn es nicht das Register-Fenster ist). Wenn die angegebenen Daten gefunden wurden, werden die Startadressen von manchen Fenstern auf die Adresse der gefundenen Daten gesetzt. Wenn innerhalb von Fenster 2 oder 3 gesucht wurde, werden die Startadressen von 2 und 3 auf die gefundenen Daten gesetzt. Wenn innerhalb der Fenster 4 oder 5 gesucht wurde, werden die Startadressen von 4 und 5 auf die gefundenen Daten gesetzt, außer dann, wenn Fenster 4 als Source-Fenster benutzt wird.

Im Source-Fenster suchen

Wenn Sie den G-Befehl im Source-Fenster verwenden, kann nur nach Text gesucht werden. Wenn er gefunden worden ist, wird die Zeile, die den Text enthält, angezeigt. Wenn der Text nicht gefunden wurde, wird das Fenster nicht verändert. Daß die Suche ohne Ergebnis verlief, erkennen Sie daran, daß die Meldung »Suche...« nicht mehr im Mitteilungsfenster unten zu sehen ist.

4.13.11 Verschiedenes

Ctrl-P Voreinstellungen

Mit dieser Dialogbox können Sie verschiedene Optionen von MonAm einstellen. Die ersten drei Fragen verlangen J/N-Antworten. Sie können, wie in jeder Dialogbox, mit **Esc** abbrechen und mit **Return** die neuen Einstellungen übernehmen.

Relative Offsets

Diese Einstellung ist normalerweise eingeschaltet und beeinflußt die symbolische Disassemblierung vom Adreßregister indirekt mit dem Offset-Adreßmodus, d.h. xxx(An). Wenn die Option eingeschaltet ist, wird jeder derartige Adreßmodus ausgerechnet und dann die Symboltabelle durchsucht; wenn ein Symbol sich an dieser Adresse befindet, wird der Befehl als Symbol(An) disassembliert. Diese Option ist sehr hilfreich, sowohl bei manchen Assemblerprogrammstilen als auch bei der Untersuchung von Hochsprachen, die ein Adreßregister als Basisregister benutzen. HiSoft Basic z.B. verwendet A3 als Zeiger auf das Runtime-System.

Symbol-Optionen

Diese Option ermöglicht es, die Behandlung von Symbolen durch MonAm zu verändern. Zuerst werden Sie gefragt, wie viele Buchstaben in einem Symbol als signifikant gelten sollen (minimal 8, maximal 22) und danach, ob die Symbole case-sensitiv verarbeitet werden sollen. Drücken von »Y« veranlaßt eine nicht-case-sensitive Suche. Diese Option ist für diejenigen gedacht, die nicht jedesmal den ganzen Symbolnamen eingeben und nicht auf Groß- und Kleinschreibung achten wollen.

Oberer RAM-Bereich

Dadurch wird MonAm angezeigt, welche Adresse im Speicher-Durchsuchen-Kommando (G) als oberer Speicherbereich angesehen werden soll. Normalerweise brauchen Sie hier keine Änderung vorzunehmen, da die Grundeinstellung 2 Mbyte beträgt, jedoch ist eine Änderung erforderlich, wenn Sie mehr Speicher benötigen sollten.

Drucker

Damit können Sie die von Ihrem Drucker benutzte Schnittstelle angeben. Der Defaultwert ist PRT:.

Voreinstellungen speichern

Antworten Sie mit *J*, wenn Sie die aktuellen Voreinstellungen in MONAM2.INF speichern wollen. Während MonAm geladen wird, werden die Voreinstellungen eingelesen. MONAM2.INF muß sich im aktuellen Verzeichnis befinden.

I Intelligentes Kopieren

Dieser Befehl kopiert einen Speicherbereich in einen anderen. Die Adressen sollten folgendermaßen eingegeben werden:

<start>,<inclusive_end>,<destination>

Die Kopie ist insofern intelligent, als der Speicherblock auch an eine solche Stelle kopiert werden kann, die sich mit der vorherigen Speicheradresse überschneidet. Die Korrektheit der Kopie wird nicht überprüft; das Kopieren von nichtexistierenden Speicherbereichen läßt MonAm abstürzen.

L Labels auflisten

Ein großes Fenster wird geöffnet, in dem alle geladenen Symbole angezeigt werden. Durch Eingabe einer beliebigen Taste gelangen Sie auf die nächste Seite, durch Esc wird die Anzeige abgebrochen. Die Reihenfolge der Anzeige entspricht der Reihenfolge, in der die Symbole auf Diskette gefunden werden.

Ctrl-U Name Deallokation des Symbolspeichers

Dieses Kommando kann nur dann angewendet werden, wenn Sie ein Programm debuggen, mit dem eine Symboltabelle geladen wurde. Mit diesem Befehl wird der Speicherplatz deallokiert, der für das Symbol verwendet wurde, und somit zum weiteren Gebrauch freigegeben. Dieser Befehl ist dann nützlich, wenn während des Debuggens eines umfangreichen Programms der Speicherplatz eng wird. Laden Sie das Programm mit den Symbolen, setzen Sie einen Breakpoint auf eine symbolische Adresse, und deallokieren Sie dann die Labels. Wenn der Breakpoint einmal gesetzt worden ist, sind alle Symbole verloren.

W Speicher füllen

Dieser Befehl füllt einen Speicherbereich mit dem angegebenen Wert. Die Parameter sind:

<start>,<inclusive_end>,<fillbyte>

Auch hier gilt der Hinweis darauf, daß keine Überprüfung stattfindet.

P Disassemblieren zum Drucker/zur Diskette

Mit diesem Befehl können Sie einen Speicherbereich zum Drucker oder auf Diskette, sowohl mit vorhandenen Symbolen als auch mit Cross-reference-Labels, disassemblieren. Die erste Zeile der Dialogbox sollte mit

<Startadresse>,<Endadresse>

angegeben werden. Die nächste Zeile fragt nach einem Speicherbereich, den MonAm zur Erstellung der Cross-reference-Liste benutzen kann:

<Pufferanfang><Pufferende>

Wenn Sie keinen wollen, geben Sie einfach eine leere Zeile ein. Die nächste Zeile fragt nach den Speicherbereichen, die als DC-Direktiven disassembliert werden sollen. Die Zeile sollte so aussehen:

<Datenanfang><Datenende>[,<Größe>]

Die optionale Größe ist die der DCs und kann B, W oder L sein, wobei L der Standardwert ist. Wenn alle Datenbereiche definiert wurden, geben Sie eine leere Zeile ein. Zuletzt wird nach einem Dateinamen gefragt. Wenn keiner angegeben wird, geht die Ausgabe zum Drucker. Wenn eine Cross-reference-Liste angefertigt werden soll, gibt es zuerst während der Erstellung eine kleine Pause. Die Labels, die so generiert wurden, haben das Format Lxxxxxx, wobei xxxxxx die Hex-Adresse des Labels ist.

Druckerausgabe

Das Zeilenformat enthält zuerst eine 8stellige Hex-Zahl, bis zu 10 Worten Hex-Daten, ggf. maximal 12 Zeichen eines Symbols, und darauf den disassemblierten Befehl. Die Druckerausgabe kann mit Esc abgebrochen werden.

Diskettenausgabe

Das Format kann direkt von GenAm verarbeitet werden. Es enthält zuerst gegebenenfalls ein Symbol, dann den disassemblierten Befehl, dem ein Tab vorausgeht. Ein Tab trennt auch den Opcode vom Operanden. Wenn Sie einen Speicherbereich ohne Symbole disassemblieren, ist es ratsam, die XREF-Option zu benutzen, da andernfalls keine Labels vorhanden sein würden. Ein Diskettenfehler oder Esc brechen die Ausgabe ab.

M Adresse verändern

Dieser Befehl ist aus Kompatibilitätsgründen zu MonAm1 beibehalten worden. Er entspricht A-A.

O Auswerten

Dieser Befehl entspricht **A-O** und ist aus Kompatibilitätsgründen zu MonAm1 beibehalten worden.

D Laufwerk und Pfad verändern

Mit diesem Befehl können Sie das aktuelle Laufwerk und den aktuellen Pfad verändern.

4.13.12 Zusammenfassung der MonAm-Kommandos

Fenster-Befehle

A-A	Startadresse setzen
A-B	Breakpoint setzen
A-E	Fenster editieren
A-F	Fontgröße
A-L	Fenster binden
A-O	Ausdruck auswerten
A-P	Drucker-Dump
A-R	Register setzen
A-S	Fenster spalten
A-T	Fenstertyp verändern
A-Z	Fenster zoomen

Bildschirmumschaltung

Breakpoints

A-B	Breakpoint setzen
Help	Breakpoint- und Segmentanzeigen
Ctrl-B	Breakpoint setzen
U	Go Until
Ctrl-K	Alle Breakpoints löschen
Ctrl-A	Breakpoint setzen und ausführen
Ctrl-X	Ausführung des Programms anhalten

Laden und Abspeichern

Ctrl-L	Ausführbares Programm laden
B	Binärdatei laden
S	Binärdatei abspeichern
A	ASCII-Datei laden und Programme ausführen

Programmausführung

Ctrl-R	Zurück zum Programm / Run
Ctrl-Z	Single-Step
Ctrl-Y	Single-Step
Ctrl-T	Instruktion interpretieren (Trace)
R	Run (verschiedene Arten)

Speicher durchsuchen

G	Durchsuche Speicher
N	Nächstes Vorkommnis

Verschiedenes

Ctrl-C	Abbrechen
Ctrl-Q	Programm verlassen
Ctrl-P	Voreinstellungen
I	Intelligentes Kopieren
W	Speicher füllen
L	Labels anzeigen
Ctrl-U	Symbole deallokieren
P	An Drucker/Diskette disassemblieren
M	Adresse verändern
O	Auswerten
D	Laufwerk und Pfad verändern
H	History-Speicher anzeigen

4.14 Fehlersuche

4.14.1 Beschränkungen

MonAm ist beim Ablauf abhängig von der Exec-, Intuition- und Grafik-Bibliothek. Wenn Ihr Programm illegalerweise Speicher zerstört, kann es möglich sein, daß dadurch unwiderruflich etwas verlorengeht, was das System unbedingt braucht. Die Maschine stürzt dann ab, wenn MonAm nach einer Exception aufgerufen wird. Dieser Fehler ist jedoch selten, normalerweise treten Adreßfehler auf, bevor ein Programm Speicherplatz zerstört. Wenn ein Programm von MonAm aus aufgerufen wird, sieht es aus, als ob es vom CLI – und nicht von der Workbench aus – gestartet worden wäre.

Im Supervisor-Modus kann MonAm jeden Code single-steppen oder darin integrierte Breakpoints bearbeiten. Die Ursache dafür ist, daß der Exec-Exception-Handler im Supervisor-Modus auf eine Exception hin überprüft und, falls dies der Fall sein sollte, eine Guru-Meldung ausgibt. Ansonsten geht er in MonAm und arbeitet normal weiter. Wenn Ihr Programm ein weiteres Programm oder eine weitere Task erzeugt, kann MonAm dieses nicht single-steppen oder Breakpoints darin abarbeiten. MonAm kann nur das Programm debuggen, das anfänglich geladen wurde.

Versuchen Sie nicht, in MonAm Standard-System-Programme wie z.B. »dir« laufen zu lassen, da diese Programme auf nicht dokumentierten Registern (besonders A5) und auf Speicherbereichen basieren, die MonAm nicht emulieren kann. Ein Wort- oder Langwortzugriff auf einen ungeraden Speicherplatz von 1 bis 7 verursacht einen kompletten Hardware-Absturz. Leider gibt es nichts, was man dagegen unternehmen kann.

4.14.2 Die Bug-Jagd

Es gibt wahrscheinlich genauso viele Bug-Jagd-Strategien, wie es Programmierer gibt, aber die Erfahrung ist doch der beste Lehrer. Wir haben in der Zeit, in der wir den 68000er programmierten, gelernt, wie man die Suche am besten angeht.

Zunächst ist es ein sehr guter Weg, sich den Quelltext anzusehen, um Fehler zu finden. Es ist eine schlechte Angewohnheit, zuerst zum Debugger zu greifen. Es könnte sein, daß Sie dabei das System oder die Programmierungsumgebung wechseln und nicht mehr auf das Low-level-Debuggen zurückgreifen können.

Wenn ein Programm offensichtliche Fehler macht, wie z.B. einen Adreßfehler, ist es wesentlich leichter, das Problem zu finden, als wenn das Programm nur manchmal, von Ihrer Sicht aus völlig unberechenbar, das falsche Ergebnis liefert.

Viele Fehler entstehen dadurch, daß bestimmte Speicherbereiche, die wichtige Daten enthalten, versehentlich überschrieben werden. Wenn man den Speicherbereich leicht erkennen kann, wie z.B. bei einem Busfehler, kann man bedingte Breakpoints an verschiedenen Stellen in Ihrem Programm unterbringen, um so den Fehler so früh wie möglich abzufangen. Nehmen wir z.B. an, daß die globale Variable »main_ptr« während der Ausführung ungerade wird. Der bedingte Ausdruck sollte so aussehen:

```
{main_ptr}&1
```

Zählerbreakpoints sind eine gute Methode dafür, Bugs abzufangen, bevor sie auftreten. Nehmen wir an, daß Sie von einer bestimmten Subroutine wissen, daß sie abstürzt, Sie den Fehler aber nicht kennen. Setzen Sie also einen Breakpoint auf sie

und lassen Sie das Programm laufen. An der Stelle, an der das Programm wegen z.B. einer Exception unterbrochen wird, sollten Sie sich mit `[Help]` den Wert des Zähler-Breakpoints ansehen. Beenden Sie das Programm, laden Sie es wieder, setzen Sie dann einen Stop-Breakpoint auf die spezielle Subroutine für den speziellen Wert oder zumindest einen vor sie. Lassen Sie das Programm dann ablaufen. Sie können so den Ablauf durch die Subroutine verfolgen. Viel Glück!

4.14.3 Exception-Analyse

Wenn eine unerwartete Exception auftaucht, ist es ganz nützlich zu wissen, warum und wo sie geschah, um dann Gegenmaßnahmen zu ergreifen.

Busfehler

Wenn der PC sich in nichtexistentem Speicher befindet, schauen Sie sich den Stack an und versuchen Sie, eine Rückkehradresse zu finden. Diese gibt meist Aufschluß über den Grund des jetzigen PC-Wertes. Wenn der PC sich innerhalb Ihres Programms befindet, hat vermutlich ein Zugriff auf einen nichtexistenten oder geschützten Speicher den Busfehler ausgelöst. Es ist sehr unwahrscheinlich, daß Sie nach einem Busfehler Ihr Programm weiterlaufen lassen können.

Adreßfehler

Wenn der PC sich nicht in Ihrem Programm befindet, ist die soeben erwähnte Methode anzuwenden. Ein Adreßfehler hat als Ursache meist einen Wort- oder Langwortzugriff auf eine ungerade Adresse. Eine Registerkorrektur hilft oft, daß das Programm wenigstens für eine Weile weiterlaufen kann.

Illegaler Befehl

Wenn der PC in einem sehr niedrigen Speicherbereich ist, etwa unter \$30, ist meist ein Sprung an die Adresse 0 schuld. Wenn Sie sich mit `MonAm` diesen Bereich ansehen, erkennen Sie mehrere ORI-Befehle, die in Wirklichkeit Langwortzeiger sind, nicht gefolgt von einem illegalen Befehl.

Privilegverletzung

Dies wird durch die Ausführung eines privilegierten Befehls im User-Modus verursacht. Meistens bedeutet dies, daß Ihr Programm Amok gelaufen ist.

Anhang

A

AmigaDOS- Fehlernummern

Dargestellt sind die Fehlernummern, die teilweise erscheinenden englischen Texte, deren Übersetzung, und, wenn nötig, zusätzliche Erklärungen.

- 103 insufficient free store
 nicht genug freier Speicher
- 104 task table full
 Task-Tabelle voll (mehr als 20 CLIs)
- 120 argument line invalid or too long
 Argument-Zeile ungültig oder zu lang (bei CLI-Kommandos)
- 121 file is not an object module
 File ist kein Objekt-Modul (kann nicht ausgeführt werden)
- 122 invalid resident library during load
 ungültige residente Library während des Ladens festgestellt
- 202 object in use
 Objekt in Gebrauch (z.B. File durch anderes Programm)
- 203 object already exists
 Objekt existiert bereits
- 204 directory not found
 Directory nicht gefunden
- 205 object not found
 Objekt nicht gefunden (File nicht gefunden)
- 206 invalid window
 ungültige Window-Spezifikation (CON:/RAW:-Window)
- 209 packet request type unknown
 Typ unbekannt

- 210 invalid stream component name
Name ist zu lang oder enthält [Ctrl]-Zeichen
- 211 invalid object lock
Ungültiges Lock
- 212 object not of required type
Objekt ist vom falschen Typ
- 213 disk not validated
Diskette wird noch geprüft oder ist defekt
- 214 disk write protected
Diskette ist schreibgeschützt
- 215 rename across device attempted
Bei »Rename« verschiedene Laufwerke angegeben
- 216 directory not empty
Directory nicht leer (muß es vor dem Löschen sein)
- 218 device not mounted
Device (Laufwerk) nicht angemeldet
- 219 seek error
Fehler beim Suchen eines Records
- 220 comment too big
File-Kommentar zu groß (>80 Zeichen)
- 221 disk full
Diskette voll
- 222 file is protected from deletion
File ist gegen Löschen geschützt
- 223 file is protected from writing
File ist schreibgeschützt
- 224 file is protected from reading
File ist lesegeschützt
- 225 not a DOS disk
keine DOS-Diskette
- 226 no disk in drive
Keine Diskette im Laufwerk
- 232 no more entries in directory
Keinen Directory-Eintrag (File-Namen) mehr gefunden

Anhang

B

GenAm

GenAm kann eine große Zahl von Fehlermeldungen ausgeben. Die meisten davon erklären sich von selbst. Dieser Anhang bringt sie in der numerischen Reihenfolge in Englisch und Deutsch, gegebenenfalls mit Erklärungen.

Bitte beachten Sie, daß GenAm laufend erweitert und noch verbessert wird. Deshalb kann diese Liste nicht genau mit Ihrer Version übereinstimmen, weil neue Meldungen hinzugekommen sind.

B.1 Fehlermeldungen

Wenn Sie eine mit INTERNAL beginnende Meldung erhalten – was niemals vorkommen sollte –, so teilen Sie uns das bitte mit.

Meldung

Bemerkung

.W or .L expected as index size

.W oder .L werden als Indexgröße erwartet

absolute expression MUST evaluate

Absoluter Ausdruck muß lösbar sein

absolute not allowed

Absoluter Ausdruck nicht erlaubt

additional symbol on pass 2

Zusätzliches Label in Paß 2

Es erschien ein Label in Paß 2,
nicht aber in Paß 1

Meldung	Bemerkung
address register expected Adreßregister erwartet	
addressing mode not allowed Adressierungsart nicht erlaubt	
addressing mode not recognized Adressierungsart nicht bekannt	
BSS or OFFSET section cannot contain data BSS- oder OFFSET-Sektion kann keine Daten enthalten	OFFSET- und BSS-Sektionen können nur DS-Direktiven enthalten
cannot create binary file Kann keine Binärdatei erzeugen	Falscher Dateiname, schreib- geschützte Diskette, o.ä.
cannot export symbol Kann Symbol nicht exportieren	
cannot import symbol Kann Symbol nicht importieren	
cannot nest MACRO definitions or define in REPTs Kann Makro-Definitionen in REPTs nicht verschachteln oder definieren	Makro-Definitionen dürfen in Wiederholungsschleifen nicht verschachtelt oder definiert werden
cannot nest repeat loops Kann Wiederholungsschleifen nicht verschachteln	
comma expected Komma erwartet	
data register expected Datenregister erwartet	

Meldung	Bemerkung
data too large Daten zu groß	
division by zero Division durch Null	
duplicate MODULE name Modulnamen kopieren	
module names must be unique Modulnamen müssen einheitlich sein	
error during listing output Fehler während der Listing-Ausgabe	Listing wird an dieser Stelle abgebrochen
error during writing binary file Fehler während des Schreibens der Binärdatei	Normalerweise Diskette voll
executable code only Nur ausführbarer Code	In den Speicher kann nur ausführ- barer Code assembliert werden
expression mismatch Fehler im Ausdruck	Normalerweise ein Syntaxerror in einem Ausdruck
fatally bad conditional Fataler Fehler in Bedingung	In einem Makro wurden mehr ENDCs als IFs verwendet
file not found Datei nicht gefunden	
forward reference Vorwärts-Referenz	
garbage following instruction Nach Befehlswort folgt Unsinn	

Meldung	Bemerkung
illegal BSR.S Illegaler BSR.S	Ein BSR.S ist für die folgende Anweisung nicht erlaubt – in BSR umändern
illegal type combination Illegale Typkombination	
immediate data expected Konstante erwartet	
imported label not allowed Importiertes Label nicht erlaubt	
include file read error Datei-Lesefehler enthalten	
instruction not recognized Unbekannter Befehl	
invalid FORMAT parameter Ungültige Format-Parameter	
invalid INCDIR Ungültige INCDIR	Sie haben mehr als 500 Byte für Verzeichnisangaben verbraucht
invalid IF expression, ignored Fehlerhafter IF-Ausdruck, wird ignoriert	
invalid MOVEP addressing mode Fehlerhafte MOVEP-Adressierungsart	
invalid number Ungültige Zahl	
invalid numeric expansion Ungültige numerische Ausweitung	Das Symbol ist nicht definiert oder relativ bzw. liegt ein Syntax-Error vor

Meldung	Bemerkung
invalid option Ungültige Option	
invalid printer parameter Ungültige Drucker-Parameter	
invalid register list Ungültige Register-Liste	
invalid section name, TEXT assumed Ungültiger Sektionsname, Text wird angenommen	
invalid size Ungültige Größe	
line malformed Zeile schlecht formuliert	
linker format restriction Einschränkung im Linker-Format	Das AmigaDOS-Format unterliegt Einschränkungen bezüglich auf erlaubte Importe. Vgl. Sektion 3.10.2
local not allowed Lokale Adressierung nicht erlaubt	
missing close bracket Fehlende Schließ-Klammer	
missing ENDC Fehlendes ENDC	Es wurden mehr IFs als ENDCs verwendet
missing quote Fehlendes Anführungszeichen	
misuse of label Fehlerhafter Gebrauch von Label	

Meldung	Bemerkung
not yet implemented Noch nicht implementiert	
number too large Zahl zu groß	
odd address Ungerade Adresse	
option must be at start Option muß am Anfang stehen	
ORG not allowed ORG nicht erlaubt	
out of memory Kein Speicher mehr	
phasing error Phasenfehler	Sollte niemals passieren
program buffer full Programmpuffer voll	Wenn in den Speicher assembliert wird, Größe des Programmpuffers verändern
register expected Register erwartet	
relative not allowed Relative Adressierung nicht erlaubt	
relocation not allowed Reloizierung nicht erlaubt	
repeated include file Wiederholte Include-Datei	

Meldung	Bemerkung
each include file may only be included once on each pass Jede Include-Datei darf nur einmal pro Paß eingebunden werden	
source expired prematurely Source-Überfluß	Innerhalb eines IF, MACRO oder REPT fließt Source über
spurious ENDC Falsches ENDC	
spurious ENDM or MEXIT Falsches ENDM oder MEXIT	
spurious ENDR falsches ENDR	
symbol defined twice Symbol zweimal definiert	
symbol expected Symbol erwartet	
undefined symbol Undefiniertes Symbol	
user Error Benutzer-Fehler	Durch eine falsche Direktive verursacht
wrong processor Falscher Prozessor	
XREFs not allowed within brackets In Klammern keine XREFs erlaubt	

B.2 Warnungen

Meldung	Bemerkung
68010 instruction, converted to MOVE SR 68010-Instruktion, zu MOVE SR konvertiert	MOVE CCR ist keine 68000-Instruktion
branch made short Branch-Verkürzung	durch Optimierung
directive ignored Direktive wird ignoriert	
invalid LINK displacement Ungültige LINK-Anzeige	Wenn größer als 0 oder ungerade
offset removed Offset entfernt	xx(An) wurde durch Optimierung in die Form (An) reduziert
relative cannot be relocated Relative Adresse kann nicht reloziert werden	
short branch converted to NOP Kurzer Sprung wird zu NOP	
sign extended operand Operanden kennzeichnen	MOVEQ benötigt Vorzeichen bei Daten
size should be .W Größe sollte .W betragen	

Anhang

C

Aufruf von System-Routinen

C.1 Einführung

Das Amiga-Betriebssystem ist mit Sicherheit eines der hochentwickeltsten aller Computer in Massenproduktion, aber auch das komplizierteste. Das ganze System basiert auf dem Libraries-Konzept, die im Prinzip Gruppen von Unterprogrammen (Funktionen für C-Programmierer) sind und über eine Indextabelle aufgerufen werden. Dieser Anhang will Ihnen grundsätzlich erklären, wie man Library-Routinen von Assembler aus aufruft, und soll Ihnen eine Vorstellung davon geben, wofür welche Routinen eingesetzt werden können. Ein kleiner Anhang kann nicht das ganze Betriebssystem erklären, er ist nur als Einführung gedacht. Für weitergehende Informationen sind die Bücher in den Literaturhinweisen zu empfehlen.

C.2 Libraries

Die wichtigste Library ist die Exec-Library, die unter anderem auch dafür gebraucht wird, andere Libraries zu öffnen. Wie für alle Libraries benötigt man einen Library-Basiszeiger, der in das Register A6 geladen werden muß, bevor man eine Routine der Lib aufrufen kann. Nur die Exec-Lib muß nicht geöffnet werden, um einen Basiszeiger zu erhalten; dieser Zeiger steht im Langwort ab Adresse 4. Das ist die einzige Adresse, für die garantiert ist, daß sie sich nicht ändert. Der daraus gelesene Basiszeiger kann benutzt werden, um andere Libs zu öffnen und damit deren Basiszeiger zu erhalten und so weiter. Beachten Sie, daß die meisten Libs den Basiszeiger in A6 erwarten, wenn sie korrekt funktionieren sollen. Das liegt daran, daß viele Routinen selbst wieder Routinen derselben Lib aufrufen und dabei unterstellen, daß der Basiszeiger schon in A6 ist. Parameter werden an Library-Routinen in Registern übergeben, wobei als allgemeine Regel gilt, daß d0/d1 und a0/a1 nach jedem Aufruf verändert sein können. Die Hauptausnahme von dieser Regel macht die Graphics-Library, worauf später noch eingegangen wird.

Mit Devpac wird eine große Anzahl von Include-Files geliefert, um einen einfachen Zugriff auf die verschiedenen Teile des Betriebssystems zu ermöglichen. Diese Include-Files enthalten Makro-Definitionen, Symbole für die Library-Offsets, Definitionen von Datenstrukturen und Symbole für Bit-Felder. Es folgt nun eine Tabelle, die die Namen der verschiedenen Komponenten zeigt und wo diese im Include-Directory zu finden sind. Darin bedeuten:

File: File mit den Makro-Definitionen und den `_LVO`-Offsets zum Aufruf der Library-Routinen.

Library	File	Name macro
	Base pointer	Calling macro
diskfont	libraries/diskfont_lib.i _DiskfontBase	DISKFONTNAME CALLDISKFONT
dos	libraries/dos_lib.i _DOSBase	DOSNAME ¹ CALLDOS
exec	exec/exec_lib.i _SysBase	EXECNAME CALLEXEC
expansion	libraries/expansion_lib.i _ExpansionBase	EXPANSIONNAME ² CALLEXP
graphics	graphics/graphics_lib.i _GfxBase	GRAFNAME CALLGRAF
icon	workbench/icon_lib.i _IconBase	ICONNAME CALLICON
intuition	intuition/intuition_lib.i _IntuitionBase	INTNAME CALLINT
mathffp	math/mathffp_lib.i _MathBase	FFPNAME CALLFFP
mathdouble	math/mathieeedoubbas_lib.i _MathIeeeDoubBasBase	IEEEDOUBNAME CALLIEEEDOUB
mathtrans	math/mathtrans_lib.i _MathTransBase	MATHTRANSNAME CALLMATHTRANS
translator	libraries/translator_lib.i _TranslatorBase	TRANSNAME CALLTRANS

Name macro: Dieser Makro besteht normalerweise aus einer DC.B-Anweisung für den Namen (in ASCII) gefolgt von einem Null-Byte.

Base pointer: Dies ist die symbolische Adresse (Label), auf der der Basiszeiger abgelegt werden sollte. Das Symbol beginnt immer mit einem Unterstrich (`_`), auch wenn die meisten C-Compiler dieses Zeichen nicht listen.

Calling macro: Dies ist der Name des Makros, mit dem eine bestimmte Library-Routine aufgerufen wird. Beachten Sie, daß dieses Makro A6 verändert, da es mit dem Basiszeiger geladen wird.

Neu ist: expansion in libraries/expansion_lib.i. Name: EXPANSIONNAME. Basiszeiger: _ExpansionBase. Makro: CALLEXP.

Zum Beispiel wäre der passende Aufruf für die Funktion »OpenLibrary«

```
CALLEXC OpenLibrary
```

Dieses Makro wird entwickelt als

```
move.l    (_SysBase).w,a6
jsr      _LV00openLibrary(a6)
```

C.2.1 Diskfont-Library

Dies ist die Library zum Umgang mit Fonts, die normalerweise auf der Disk sind.

Files: libraries/diskfont.i und diskfont_lib.i

C.2.2 DOS-Library

Diese ist eine der am einfachsten zu benutzenden Libraries und behandelt das File-I/O (Input/Output) zu Geräten einschließlich Disketten und der Konsole. Diese Lib hat einige kleine Besonderheiten. Bemerkenswert ist, daß Adressen in Datenregistern übergeben werden müssen und viele Zeiger vom Typ BCPL sein müssen (d.h. Adresse dividiert durch 4 auf eine Langwortgrenze justiert).

Files: libraries/dos.i, dos_lib.i und dosextens.i

C.2.3 Exec-Library

Dies ist die Library auf der untersten Ebene. Sie ist zum Beispiel zuständig für Speicherverwaltung, Library-Öffnen und Nachrichtenübergabe. Die Library darf niemals geöffnet werden, ihr Basiszeiger steht auf Adresse 4.

Files: exec/ables.i, alerts.i, devices.i, errors.i, exec.i, execbase.i, execname.i, exec_lib.i, funcdef.i, initializers.i, interrupts.i, io.i, libraries.i, lists.i, memory.i, nodes.i, ports.i, resident.i, strings.i, tasks.i und types.i

C.2.4 Graphics-Library

Diese Lib ist für alles zuständig, was auf Ihrem Monitor erscheint, zum Beispiel Zeichnen von Linien, Ausgabe von Text, Kontrolle von Rast-Ports, Sprites und Fonts. Beachten Sie bitte, daß in der Kickstart-Version 1.1 die Text-Routine Register D7 verändert. Dies kann auch Auswirkungen auf andere Routinen haben, besonders auf die Routine »DoIO« bei der Konsole-Ausgabe.

Files: graphics/clip.i, copper.i, display.i, gels.i, gfx.i, gfxbase.i, graphics_lib.i, layers.i, rastport.i, regions.i, sprite.i, text.i, view.i

C.2.5 Icon-Library

Diese Lib ist zuständig für den Umgang mit Icons, die auf der Workbench angezeigt werden.

Files: workbench/icon.i und icon_lib.i

C.2.6 Intuition-Library

Diese Library ist die größte und zuständig für die Bedienungsoberfläche Intuition. Sie verfügt über eine sehr große Anzahl von Funktionen wie die Kontrolle von Windows, Screens, Gadgets, Requestern und dem Event-Handling. Das Hauptfile ist groß und lädt mit Include noch andere Files nach. Seien Sie nicht überrascht, wenn es eine Weile dauert, bis alles geladen ist! Es kann sich durchaus lohnen, eine eigene Version zu erstellen, in der die weniger oft benutzten Konstanten und Include-Files nicht enthalten sind.

Hinweis des Übersetzers: Schauen Sie sich die Include-Files an. Es ist oft sehr wenig, was da »included« wird und lohnt kaum den Disk-Zugriff. Andererseits werden oft große Files nachgeladen, obwohl man daraus nur eine oder zwei Konstanten braucht.

Files: intuition/intuition.i und intuition_lib.i

C.2.7 Maths-Libraries

Es gibt drei Mathematik-Libraries, die alle auf den offiziellen Motorola-Routinen basieren. Die FFP- (Fast Floating Point = schnelles Fließkomma) Library rechnet mit einem 8-Bit-Exponenten und einer 24-Bit-Mantisse. Das Format wurde von Motorola für die 68000er entwickelt und ist kein Standardformat. Die »IEEE double«-Library bietet doppelte Genauigkeit im IEEE-Format und die »Transzendenten«-Lib trigonometrische und andere Funktionen im FFP-Format.

Files: math/mathffp_lib.i, mathieeedoubbas_lib.i und mathtrans_lib.i

Generell sollten Sie GenAmiga im case-sensitiven Modus (wie voreingestellt) benutzen, wenn Sie die mitgelieferten Include-Files einsetzen. Beachten Sie, daß jedes Include-File automatisch alle Include-Files lädt, die es benötigt, so daß Sie sich darum nicht kümmern müssen.

C.3 Beispielprogramme

Um Ihnen den Anfang in der Programmierung von AmigaDOS in Assembler zu erleichtern, haben wir einige Beispielprogramme im Directory »example« vorgesehen:

C.3.1 demo.s

Dies ist das Programm, das in der Einführung zu Beginn dieses Handbuches benutzt wurde. Es nutzt die DOS-Library und gibt einen Text im aktuellen CLI-Fenster aus.

C.3.2 freemem.s

Dieses Programm unter Intuition öffnet ein Fenster, in dem der freie Hauptspeicher ständig angezeigt wird, bis das Schließ-Gadget angeklickt wird. Um das Programm als zum CLI parallele Task laufen zu lassen, geben Sie das CLI-Kommando »run freemem« ein.

C.3.3 helloworld.s

Dies ist die Assembler-Umsetzung des C-Programms (letzte Version) »Simple Program«. Die Umsetzung wurde in keiner Weise optimiert. So ist es zum Beispiel wesentlich effektiver, die Zuweisungen für »NewScreen« durch DC-Befehle zu ersetzen. Das Programm öffnet einen Screen, darauf ein Window und druckt dann einen Text.

C.4 Vergleich CLI / Workbench

Es gibt zwei Programmumgebungen auf dem Amiga, die Window- und Icon-gesteuerte Workbench und das CLI. Devpac-Amiga selbst läuft unter letzterem, wie auch die meisten Beispielprogramme. Der Unterschied liegt im Startup- und Exit-Code.

C.4.1 CLI-Start

Wenn ein Programm vom CLI aus gestartet wird, enthält Register A0 die Adresse der Kommandozeile und D0 deren Länge. Die DOS-Handles, die die Funktionen »Input« und »Output« zurückgeben, können für die Ein-/Ausgabe vom bzw. zum aktuellen CLI-Fenster benutzt werden. Solche Programme enden mit einem einfachen RTS.

C.4.2 Workbench-Start

Wenn ein Programm von der Workbench gestartet wurde, muß es zuerst auf eine Nachricht warten und diese Nachricht (nach einem Forbid-Aufruf) zurückgeben, bevor es mit RTS endet. Die DOS-Funktionen Input und Output geben ungültige Handles zurück, weshalb Sie ein eigenes Fenster für die Ein-/Ausgabe öffnen müssen.

Eine abgemagerte Version dieses Programms für Assembler-Programmierer finden Sie im File »misc/easystart.i«. Dieses File wird auch vom Beispielprogramm »freemem2.s« eingezogen. Das Include-File sollte ziemlich zu Beginn Ihres Programms eingefügt werden. Es erledigt die Nachrichtenübergabe so, daß das Programm von der Workbench aus gestartet werden kann. Natürlich benötigen Sie dazu ein Icon, das mit IconEd erstellt werden kann.

C.5 Andere 68000-Prozessoren

Wenn Sie kommerzielle Programme für den Amiga schreiben, sollten Sie bedenken, daß einige Anwender einen 68010 oder 68020 anstatt des üblichen 68000 einsetzen. Das einzige Problem dabei ist der »MOVE SR«-Befehl, der beim 68010 und 68020 privilegiert ist. Zur Lösung des Problems bietet die Exec-Library die Routine GetCC, welche die Condition-Codes im Register D0 zurückgibt. Sie gebraucht keine anderen Register und braucht (so der Ist-Stand) auch nicht den Library-Basiszeiger in A6. Um zum Beispiel die Condition-Codes auf den Stack zu bringen, schreiben Sie anstatt

```
move      sr,-(sp)
```

besser

movea.l	4.w,a0	hole Basiszeiger
jsr	_LVOGetCC	Aufruf von GetCC
move.w	d0,-(sp)	CCs auf den Stack
move	d0,ccr	und Codes setzen

Die Routine nutzt »A0« als Arbeitsregister. Am besten schreiben Sie das als Makro, um ein versehentliches Ändern des Condition-Code zu vermeiden (»movea« ändert sie nicht).

Der einzige weitere Unterschied ist der wesentlich größere Stackbereich, den die 68010/68020-Prozessoren im Falle von Exceptions füllen. Doch das dürfte nur Debugger und ähnliche Programme betreffen.

Anhang

D

Gebrauch des CLI

D.1 Einführung

Wie die meisten Programmmentwicklungs-Tools ist auch Devpac Amiga für den Einsatz unter dem CLI (Command Line Interface) vorgesehen. Wenn Sie sich die Diskette von der Workbench aus ansehen, werden Sie deshalb wenige (oder keine) Icons sehen.

Devpac Amiga wird mit seiner eigens modifizierten Workbench-Diskette geliefert, die die Workbench-Bedienungsoberfläche nicht lädt, sondern direkt ins CLI geht.

D.2 Files, Laufwerke und Directories

AmigaDOS-Files können auf Diskette, der RAM-Disk oder auf einer Hard-Disk gespeichert werden. Weil Disketten 800 Kbyte Daten enthalten können und Hard-Disks sogar noch mehr, erlaubt AmigaDOS Directories. Das sind mit Namen bezeichnete Sektionen auf den Disks, die Files oder Directories enthalten können. (Wenn Sie die Workbench benutzen, heißen Directories Drawers (Schubladen).) Um ein File innerhalb eines Directories anzusprechen, wird das »/«-Zeichen benutzt. Wenn Sie also ein File mit dem Namen »test.s« editieren wollen, der im Directory »source« steht, gilt das Kommando

```
genam2 source/test.s 
```

Beachten Sie, daß das CLI Groß- und Kleinbuchstaben nicht unterscheidet, weshalb die Eingabe

```
GenAm2 Source/Test.s 
```

dasselbe bewirkt.

Der Aufruf von GenAm ist ähnlich dem der meisten CLI-Kommandos. Sie bestehen aus einem Kommando (hier GenAm), dem optional noch etwas folgt, das man Kommandozeile nennt.

Wenn Sie wissen wollen, welche Files auf der Disk sind, geben Sie

```
dir 
```

ein. Daraufhin werden alle Files gezeigt (dir heißt *zeige DIRectory*). AmigaDOS kann dafür länger brauchen als andere Systeme, seien Sie also geduldig. Wenn es Directories auf der Disk gibt, werden diese zuerst angezeigt, und zwar gefolgt von »(dir)«. Darauf folgen die File-Namen in alphabetischer Reihenfolge.

Unglücklicherweise kümmert sich AmigaDOS nicht um die Schreibweise der File-Namen hinsichtlich Groß-/Kleinbuchstaben, jedoch erfolgt die Unterscheidung im Ausdruck. Dies liegt darin begründet, daß der Name in der Schreibweise benutzt wird, der beim Anlegen des Files oder Directorys gewählt wurde.

Das CLI greift immer auf das aktuelle Laufwerk und Directory zu. Mit dem cd-Kommando können Sie den jeweiligen Stand erfragen und ändern. Nach dem Booten mit einer Kopie der Devpac-Diskette ist die aktuelle Disk DF0: (DF0: = internes, DF1: = externes Laufwerk). Sie können das testen mit

```
cd 
```

Tippt man nur »cd« ein, wird immer das aktuelle Directory angezeigt. Wenn Sie in ein anderes Directory wechseln wollen, zum Beispiel in das mit den Beispielprogrammen, müssen Sie »cd« noch den Namen folgen lassen, d.h.

```
cd examples 
```

Wenn Sie jetzt »dir« eingeben, sehen Sie eine Liste von Files. Geben Sie nun nur »cd« ein, sollten Sie »DF0:examples« sehen.

Um sich Textfiles anzusehen, die mit ».s« enden, können Sie zum Beispiel eingeben

```
type demo.s 
```

Das zeigt den Inhalt von »demo.s« auf dem Schirm. Um das (durchlaufende) Bild anzuhalten, können Sie die rechte Maustaste drücken; um das Listing vorzeitig abubrechen, geben Sie ein.

Um in das vorherige Directory zurückzukommen, geben Sie ein

```
cd df0: 
```

oder alternativ

```
cd : 
```

Der Doppelpunkt alleine bezeichnet das Directory auf der obersten Ebene (Root-Directory). Wenn wir uns nun »demo.s« noch einmal ansehen wollen, können wir wieder das Directory wechseln, aber einfacher ist es, das Directory zusammen mit dem File-Namen anzugeben, also

```
type examples/demo.s 
```

Beachten Sie die Art, wie der Schrägstrich (/) eingesetzt wird.

D.3 AmigaDOS-Wildcards

Wildcards sind Platzhalter für beliebige Zeichen oder Zeichenfolgen. AmigaDOS hat unglücklicherweise eine völlig unterschiedliche Lösung für Wildcards als andere Betriebssysteme. Zugegebenermaßen besitzt man damit viel mehr Möglichkeiten für die Arbeit, dennoch ist die AmigaDOS-Version nicht sehr einfach zu verstehen. Hier eine Zusammenfassung:

?	ein einzelnes Zeichen
%	Nullstring
#<p>	ein- oder mehrfaches Auftreten des Musters <p>
<p1> <p2>	Entweder Muster p1 oder Muster p2
()	faßt Muster zusammen

Im allgemeinen braucht man davon das »?«, das sich von selbst erklärt, und »#?«, was dem »*« von MS-DOS und CP/M entspricht.

D.4 Gerätenamen

Alle I/O-Geräte des Amiga haben Namen, über die sie in CLI-Kommandos und in Programmen angesprochen werden können. Es sind kurze Namen, die mit einem Doppelpunkt enden. Tatsächlich haben Sie schon zwei davon kennengelernt, nämlich die beiden Floppy-Laufwerke DF0: und DF1:. Hier ist der Rest:

D.4.1 RAM:

Dies ist die RAM-Disk, die wie ein normales Laufwerk eingesetzt werden kann. Sie ist allerdings sehr viel schneller, weil die Daten direkt im Hauptspeicher aufgezeichnet werden. Ihr Nachteil ist, daß die Daten verlorengehen, wenn der Rechner ausgeschaltet wird, ein Reset erfolgt oder Ihr Programm etwas verrückt spielt. Die RAM-Disk ist dynamisch, d.h. sie belegt immer nur so viel Speicher, wie für das jeweilige Datenvolumen benötigt wird.

Anmerkung zu Kickstart 1.1: Wenn man die RAM-Disk bis zum letzten Byte füllt, stürzt das System ab, weil es ohne Erfolg (mangels RAM) versucht, die Meldung »Disk voll« auszugeben.

D.4.2 RAD:

Unter der Workbench 1.3 ist RAD: eine RAM-Disk, die sich von RAM: auf zwei Arten unterscheidet:

- Ihre Größe ist auf mount-time fixiert.
- Sie überdauert einen Warmstart mit `Ctrl` - rechte `A` - linke `A` und kann mit einem Kickstart gelöscht werden.

D.4.3 PAR:

Dies ist der Parallel-Druckerport, der normalerweise nur für die Ausgabe (Output) eingesetzt wird.

D.4.4 SER:

Dies ist die serielle Schnittstelle, die für Ein- und Ausgabe benutzt werden kann, normalerweise zu Druckern und Modems.

D.4.5 PRT:

Dies ist der allgemeine Druckerport, der mit *Voreinstellungen* konfiguriert wird. Hierfür muß das Programm nicht wissen, an welchem Port (PAR: oder SER:) der Drucker hängt oder welcher Druckertyp angeschlossen ist.

D.4.6 NIL:

Dies ist ein Dummy-Gerät, auf das man Ausgaben ins Nichts schicken kann. Der Versuch von NIL: zu lesen, generiert die Fehlermeldung »end of file«.

D.4.7 CON:

Dies ist die Konsole. Damit wird ein Window für die Tastatur-Ein-/Ausgabe geöffnet. Die Syntax lautet:

CON:x/y/Breite/Höhe/Titel

D.4.8 RAW:

Das ist, um das AmigaDOS-Handbuch zu zitieren, »für den fortgeschrittenen Anwender gedacht. Experimentieren Sie nicht mit RAW:«.

D.4.9 *

Ist zwar keine Einheit, kann aber als Dateiname verwendet werden und bezeichnet das aktuelle CLI-Fenster.

D.5 AmigaDOS-Befehle

Um Ihnen beim Einsatz des CLI zu helfen, folgt nun eine kurze Beschreibung der am meisten verwendeten Befehle zusammen mit Beispielen. Es ist keine vollständige Beschreibung. Wenn Sie weitere Details benötigen, empfehlen wir, auf weiterführende Literatur zurückzugreifen.

»[]« bezeichnen erforderliche Parameter, »< >« optionale. Die meisten Kommandos erlauben die Umleitung der Ausgabe mit dem »><-Zeichen, so kann zum Beispiel ein Directory-Listing in einen File geschrieben werden mit

```
dir >allfiles.txt
```

ASSIGN <Name> <Directory>

Dies erlaubt Ihnen, spezielle Namen – besonders für Directories – zu vergeben. Beim Start werden vom System die folgenden Zuweisungen gesetzt:

Name	Voreinstellung	Anwendung/Zweck
C:	boot:c	CLI sucht hier Kommandos
L:	boot:l	System-Handlers
S:	boot:s	Startup-sequence
LIBS:	boot:libs	Libraries nicht im ROM
DEVS:	boot:devs	Geräte-Treiber
FONTS:	boot:fonts	Zeichensätze nicht im ROM
SYS:	boot:	Boot-Diskette

Genauso gut, wie Sie das ändern können, können Sie auch eigene Zuweisungen machen. Wenn Sie zum Beispiel einfacher auf das Graphics-Include-File auf der Diskette im externen Laufwerk zugreifen wollen, schreiben Sie

```
assign graf: df1:include/graphics 
```

Dann können Sie unabhängig vom aktuellen Directory auf Files in dieser Art zugreifen:

```
type graf:rastport.i
```

Um alle Zuweisungen aufzulisten, geben Sie nur »assign« ein, und um eine Zuweisung aufzuheben, wiederholen Sie sie, ohne das Directory (den Pfad) anzugeben.

CD <Directory>

Dies ändert entweder das Directory, wenn dem CD ein Parameter folgt, oder zeigt das aktuelle Directory an (wenn nur CD eingegeben wird).

COPY [Altname] <TO> [Neuname] <ALL> <QUIET>

COPY kopiert Files einzeln oder in Gruppen oder auch ein ganzes Directory. Mit der ALL-Option werden Directories kopiert. Die QUIET-Option unterdrückt das Listing der Filenamen. Jedes existierende File wird ohne Rückfrage überschrieben. Am besten zeigt man das anhand von Beispielen:

```
copy examples/demo.s to ram:demo.s 
```

kopiert ein einzelnes File.

```
copy examples to df1:genexamples all 
```

kopiert ein komplettes Directory in ein anderes (das bereits existieren muß).

```
copy include to df1:include all 
```

kopiert das Include-Directory sowie alle Directories darin auf das externe Laufwerk und legt dabei die neuen Directories an, soweit nötig.

```
copy #?.info ram:
```

kopiert alle Files, die mit ».info« enden, auf die RAM-Disk.

DATE <Datum> <Zeit>

DATE zeigt oder setzt das Datum und/oder die Uhrzeit. Das Format für das Datum ist TT-MMM-JJ, das für die Zeit HH:MM. Es werden beide zusammen oder einzeln gesetzt. Folgt kein Parameter, wird der Ist-Stand angezeigt. Beispiel:

```
date 25-jan-87 12:00
```

DELETE [Name] <Name> <Name>.. <ALL>

DELETE löscht Files einzeln oder in Gruppen oder auch ein ganzes Directory. Wenn Sie nur ein Directory angeben, wird es nur gelöscht, wenn es leer ist. Die ALL-Option löscht aber vorab den Inhalt, z.B.

```
delete demo.s 
```

```
delete examples all 
```

DIR <Name>

Listet das aktuelle Directory, wenn »Name« fehlt, sonst das benannte Directory.

DISKCOPY (Laufwerk) TO (Laufwerk) <NAME Name>

Dadurch wird eine Diskette komplett auf eine andere kopiert. Die Zieldiskette muß nicht formatiert sein, alle darauf vorhandenen Files werden gelöscht. Wenn Sie für beide Laufwerke denselben Namen angeben, werden Sie jeweils zum Diskettenwechsel aufgefordert. Es wird empfohlen, auf der Quelldiskette den Schreibschutz zu aktivieren, bevor Sie das Programm starten. Sonst könnten versehentlich Daten verlorengehen. Per Voreinstellung bekommt die Zieldiskette den Namen des Originals, das kann aber mit der NAME-Option geändert werden.

```
diskcopy df0: to df1: 
```

```
diskcopy df0: to df0: name "Backup" 
```

ECHO <String>

Das Kommando gibt »String« auf dem Schirm aus. Es ist hauptsächlich in der Startup-Sequence (siehe Abschnitt D.6) und in Execute-Batch-Files von Nutzen. Beispiel:

```
echo "Devpac Amiga Workbench disk" 
```

ENDCLI

Damit verlassen Sie das aktuelle CLI, aber Vorsicht!!! Sind Sie nämlich schon im letzten CLI (CLI 1) und haben Sie nicht die Workbench geladen (mit »loadwb«), kommen Sie nicht mehr zurück. Sie können dann nur noch neu booten. Der Befehl ist zum Verlassen von CLIs gedacht, die Sie vorher mit NEWCLI eingerichtet hatten.

EXECUTE [Name] <Argumente>

Für häufig verwendete Kommandofolgen im CLI kann man ein Textfile anlegen, in dem die Kommandos aufgeführt werden. Ruft man dieses Textfile mit EXECUTE auf, so werden alle Befehle nacheinander abgearbeitet. Parameter werden an Stelle von Platzhaltern eingesetzt. Dies funktioniert ähnlich wie bei den Makros von GenAm. Heißt das Textfile zum Beispiel »doit« und enthält diese Anweisungen

```
.key file/a
```

```
copy <file> to ram:
```

```
copy ram:<file> to prt:
```

```
delete ram:<file>
```

dann bewirkt

```
execute doit demo.s 
```

daß das File demo.s auf die RAM-Disk kopiert wird, von dort auf den Drucker (und somit gedruckt wird) und schließlich auf der RAM-Disk gelöscht wird. Das »key« spezifiziert einen Parameter (hier file) und »/a« bedeutet, daß der Parameter immer (always) erforderlich ist.

FAULT <Zahl>

Dieses Kommando gibt die Bedeutung (als Text) einer Fehlernummer von AmigaDOS auf dem Schirm aus.

FORMAT DRIVE [drivename] NAME [diskname]

FORMAT initialisiert (formatiert) eine Diskette, wobei ihr bisheriger Inhalt zerstört wird. »drivename« kann DF0: oder DF1: sein, »diskname« ist der Name der Diskette. Beispiel:

```
format drive df1: name "My backups" 
```

INFO

INFO bringt folgende Informationen auf den Schirm: Status jedes Laufwerks und jeder montierten Diskette einschließlich deren Namen und des freien Bereichs in Blöcken. Ein Block hat 512 Byte, womit die Hälfte der Blockzahl die Größe in Kbyte angibt.

JOIN [file1] <file2 file3 u.s.w.> AS [newfile]

JOIN kopiert bis zu 15 Files zusammen in ein neues File, wie zum Beispiel:

```
join part1 part2 AS allparts 
```

LIST <Name>

LIST wirkt wie DIR, nur daß auch noch die File-Größe und das Entstehungsdatum angezeigt werden.

LOADWB

Dieses Kommando lädt die Workbench und bringt deren Screen hinter das CLI-Window. Die Standard-Workbench-Disketten haben diese Anweisung in der »Startup-sequence«, so daß die Workbench automatisch geladen wird. Auf der GenAm-Diskette fehlt das, womit Speicher gespart wird.

MAKEDIR [Directory]

Damit wird ein neues Directory angelegt. Es darf kein File oder Directory gleichen Namens bereits existieren.

NEWCLI <Window>

Damit wird ein neues CLI angelegt, das als parallele Task läuft. Zusätzlich kann das Window wie bei CON: (siehe D.4.7) spezifiziert werden.

RENAME [Altfile] <TO> [Neufile]

Dieses Kommando ändert den Namen eines Files. Es kann auch benutzt werden, um ein File in ein anderes Directory zu bringen. Beispiele:

```
rename demo.s to demo.bak   
rename include/misc/start.i to examples/startup.s
```

RUN [Kommando]

RUN erlaubt, Kommandos oder Programme parallel (gleichzeitig) ablaufen zu lassen. Dazu wird ein neues CLI (ohne Fenster) geöffnet, dem das Kommando übergeben wird. Nach dessen Ende wird das CLI wieder entfernt. Wenn Sie eine Folge von Kommandos so laufen lassen wollen, müssen Sie ein Pluszeichen (+) an das Ende jeder Zeile setzen. Beispiele:

```
run copy demo.s to prt:   
druckt den File demo.s im Hintergrund, während  
run copy demo.s to prt:+   
echo "File ist gedruckt"   
das gleiche tut, dann aber noch die Meldung bringt.
```

STACK <Größe>

Dieses Kommando setzt die Stack-Größe für alle Tasks, die unter dem CLI laufen, oder zeigt nur die Ist-Größe an, wenn man »Größe« wegläßt. Voreingestellt sind 4000 Byte, was für alle CLI-Kommandos und unsere Programme ausreicht, allerdings mit einer Ausnahme. Beim DIR-Kommando reichen bei sehr tiefgeschachtelten Directories 4000 Byte nicht aus. Außerdem gibt es andere Programme (wie einige C-Compiler), die einen größeren Stack benötigen. Das sollte aber im Handbuch dieser Programme dokumentiert sein.

TYPE [Name] <OPT N> <OPT H>

Das Kommando wird normalerweise benutzt, um ein Textfile auf dem Schirm auszugeben, wobei das Display mit der rechten Maustaste angehalten werden kann. Wenn Sie ein Nicht-Textfile ausgeben, kann es vorkommen, daß der Zeichensatz umgeschaltet wird und dann nur noch Unsinn erscheint. Um das zu korrigieren, drücken Sie

-[O]

Die N-Option fügt Zeilennummern hinzu, mit der H-Option erreichen Sie den üblichen Hex-Dump mit gleichzeitiger ASCII-Darstellung (nicht druckbare Zeichen als Punkte).

WHY

WHY (warum) kann nach einer CLI-Fehlermeldung eingegeben werden und bringt dann noch weitere Erklärungen.

D.6 Startup-Sequence

Wenn Sie von einer Workbench-Diskette booten, wird das File »s/startup-sequence« geladen, dann werden alle darin aufgeführten Kommandos ausgeführt. Eines dieser Kommandos heißt »loadwb« (lade Workbench). Genau diese Anweisung fehlt auf der GenAm-Diskette, außerdem auch der sonst vorhandene ENDCLI-Befehl. Damit bleiben Sie im CLI. Wenn Sie das File editieren wollen, können Sie das mit GenAm tun. Geben Sie ein

genam s:startup-sequence

(s: bewirkt, daß das File unabhängig vom aktuellen Directory gefunden wird.) Sie können nun das File ändern. Die neue Sequenz wird beim nächsten Booten ausgeführt.

Anhang

E

Konvertierung von anderen Assemblern

Einführung

Die meisten Assembler für den Amiga folgen mehr oder weniger dem Motorola-Standard. Die Befehle an sich sind zwar am Standard ausgerichtet, die Syntaxregeln für Labels, Kommentare und Direktiven können dagegen variieren. In diesem Anhang werden die Veränderungen behandelt, die sich ergeben, wenn Programme von einem anderen Assembler konvertiert werden, gleichgültig, ob es alte Quelldateien oder in einer Zeitschrift abgedruckte Listings sind.

E.1 Amiga-Makro-Assembler und MCC-Assembler

Fast alle Quellcodes, die unter dem von Commodore unterstützten AmigaDOS-Makro-Assembler und dem Metacomco-Assembler (MCC) geschrieben wurden, können mit ein paar kleinen Änderungen unter GenAm assembliert werden. Die Unterschiede sind:

1. GenAm erzeugt Fehlermeldungen, wenn Konstanten importiert werden, die XREF verwenden, und auf die dann als Konstanten (als Gegensatz zu relativen Adressen) zugegriffen wird. Viele Quelltexte benutzen XREFs für _LVO-Labels. Um den Warnungen entgegenzuwirken, tun Sie am besten folgendes:

- Ändern Sie XREF in XREF.L.
- Verwenden Sie OPT W-, um alle Warnungen zu unterdrücken.
- Verwenden Sie OPT T-, um eine Typenüberprüfung zu vermeiden.
- Includen Sie die entsprechende _lib.i-Datei, und entfernen Sie XREF.

2. Der Befehl RORG wird nicht unterstützt.

Die mit dem AmigaDOS-Makro-Assembler mitgelieferten Include-Dateien werden von GenAm ohne Änderungen assembliert, jedoch sollte den von uns gelieferten Include-Files der Vorzug gegeben werden, da sie anstatt Makros Direktiven ver-

wenden und schneller assembliert werden. Außerdem brauchen Sie weniger Speicherplatz.

E.1.1 K-Seka

In GenAm ist nach Labels kein Komma erforderlich, jedoch benötigen Direktiven und Befehle, die im Label-Feld beginnen, einen vorangesetzten Tab. Eine Anzahl von Seka-Direktiven haben als Grundeinstellung anstatt Wortgrößen Bytes. Gleichwertige Direktiven sind:

D=DC, BLK=DS, IF=IFNE, ELSE=ELSEIF, ENDIF=ENDC.

In der Makro-Syntax müssen »?« in »\s« geändert werden; jedoch muß »?0« in »\@« geändert werden.

E.2 Neue Commodore-Include-Files

Sollten Sie neue Include-Dateien von Commodore besitzen, müßten Sie ohne Änderungen unter GenAm assemblieren können. Wollen Sie sie jedoch in das Format konvertieren, das von uns unterstützt wird, um Assemblierzeit und Speicherplatz zu sparen, benutzen Sie die von uns gelieferten Tools ConvertFD und ConvertI.

E.2.1 ConvertI

ConvertI wird auf Disk 2 ausgeliefert und konvertiert Commodore-Include-Dateien in das komprimierte Format, das wir verwenden. Die Kommandozeile sieht so aus:

```
ConvertI [Quelle] [Ziel] <-d>
```

Die Extension »i« der Quell- und Zieldatei muß weggelassen werden. Das Flag -d zeigt an, daß das ganze Directory konvertiert wird. Zum Beispiel konvertiert

```
ConvertI alt/dos neu/dos
```

die Datei alt/dos.i in neu/dos.i.

```
ConvertI alt neu -d
```

konvertiert die Standard-Include-Dateien im Verzeichnis »alt« in die jeweils korrespondierende Datei im Verzeichnis »neu«.

E.2.2 ConvertFD

ConvertFD wird auf Disk 2 mitgeliefert und konvertiert Commodore-FD-Dateien in Library-Include-Dateien mit der Extension »_lib.i«, die die _LVO-Offsets enthalten, die in Ihr Programm includet werden sollen.

Die Kommandozeile hat das Format

```
ConvertFD [Quelle [Ziel] <-d>
```

Die Extensionen der Quell- und Zielformate müssen weggelassen werden. -d zeigt an, daß ein ganzes Verzeichnis konvertiert wird. Zum Beispiel konvertiert

```
ConvertI fdfiles/dos neu/dos
```

die Datei »fdfiles/dos_lib.fd« in »neu/dos_lib.i«.

```
ConvertI fdfiles:include/libraries -d
```

konvertiert alle fd-Dateien im Verzeichnis »fdfiles« in das entsprechende Include-File im Verzeichnis »:include/libraries«.

Anhang

F

Verbesserungen in der Version 2.0

Dieser Abschnitt ist als Schnellübersicht für diejenigen Benutzer gedacht, die das Update von der Version 1.0 auf Version 2.0 erworben haben.

F.1 Der Editor

Der Editor läuft nun nach entsprechender Voreinstellung unter 60- und 80spaltigem Textmodus. Die Speichergröße kann nun im Editor eingestellt werden, wodurch das Installationsprogramm der Version 1.0 überflüssig wird.

Block löschen kann nun mit **(Shift)-(F5)** und **(Shift)-(F3)** erreicht werden. Der Block wird im Blockspeicher behalten und kann so nach Bedarf wieder eingefügt werden. Blockmarkierungen werden auf dem Bildschirm angezeigt.

F.1.1 Der Assembler

Die ersten 127 Zeichen von Symbolen sind signifikant, lokale Labels werden unterstützt; das Format entspricht dem des Amiga-Makro-Assemblers. Mit der INCBIN-Direktive wird ein Binärfile in eine Ausgabedatei kopiert, was für Bildschirmdaten sehr nützlich sein kann.

Include-Files werden nur noch einmal gelesen, wodurch Speicherplatz und Zeit gespart wird. Die maximale Assembliergeschwindigkeit beträgt 75.000 Zeilen pro Minute, der Durchschnittswert beträgt 35.000 Zeilen pro Minute. Die Geschwindigkeit, mit der Symboltabellen durchsucht und aufgeteilt werden, wurde ebenfalls erhöht.

Makro-Aufrufe und Includes können so tief verschachtelt werden, wie es der Speicherplatz erlaubt, IFs bis zu 64 Ebenen tief. Der Output von Dateinamen wird besser kontrolliert; verschiedene AmigaDOS-Sektionen (oder Module) werden nun voll unterstützt.

Makros können bis zu 36 Parameter von mehrlinigen Aufrufen und numerischer Substitution bei sich haben. Der Makro-Puffer wird dynamisch verwaltet, wobei nicht länger der freie Speicher des Editors dafür verwendet wird. REPEAT-Schleifen werden nun ermöglicht; eine nützliche Erweiterung stellen Vergleichsoperatoren dar. Mit der REG-Direktive können symbolische Registerlisten verwendet werden. Die OPT-Direktive unterstützt eine stattliche Anzahl von Extensionen. Oktale Zahlen sind inzwischen zugelassen. Register können mit R0 bis R15 aufgerufen werden.

Der Gebrauch von \W und \L nach Register-Equates, die als Index-Register verwendet werden, ist nicht länger erforderlich, da z.B. nach »buf« als Register-Equate »move.b d0,0(a6,buf.l)« nicht erlaubt ist. In MOVEMs sind Register-Equates erlaubt.

Kompatibilität

Die meisten Quelldateien sollten ohne größere Änderungen assembliert werden. Die Unterschiede sind:

- Symbole sind nun case-sensitiv, die ersten 127 Zeichen sind signifikant.
- OPT N für schmale Listings wurde durch die FORMAT-Direktive ersetzt, die eine größere Flexibilität erlaubt. Die Listing-Kontrolle wurde generell verbessert.
- In Version 1.0 wurde BRA.W nicht akzeptiert, dagegen aber BRA.L, der genau genommen ein 68029er-Befehl ist. BRA.L generiert nun eine Warnmeldung, BRA.W (und BRA.B) werden nun angenommen.
- Bezüglich des Parsing von Ausdrücken wurden mehrere kleinere Änderungen unternommen, um die Flexibilität zu erhöhen. Die Verwendung von (Ausdruck)\W (das eine Kurzwort-Adresse angibt) innerhalb eines Makros wird ignoriert, da sich \W und \L auf Makro-Parameter beziehen und durch Nullen ersetzt werden. Benutzen Sie statt dessen (Ausdruck).W, das sich im Motorola-Standard befindet.
- Die Priorität des Gleichheits-Operators (=) wurde geändert.

Der Assembler zeigt syntaktische Fehler bereits in Pass 1 an; wenn Fehler gefunden wurden, wird Pass 2 nicht gestartet. Da AmigaDOS-Output-Routinen verwendet werden, kann mit einem Tastendruck eine Pause gemacht, mit Return abgeschickt und mit Ctrl-C abgebrochen werden.

Es gibt eine Anzahl neuer Direktiven, die sich theoretisch mit Makro-Namen in GenAml-Quell-Dateien überschneiden könnten: COMMENT, DCB, ELSEIF, ENDR, FORMAT, IIF, INCBIN, OFFSET, OUTPUT, REG, REPT, RSSET, SUBTTL.

Es gibt drei reservierte Symbole, die mit einem Unterstrich beginnen: `_LK`, `_RS` und `G2`.

F.1.2 Der Debugger

MonAm unterstützt eine Anzahl neuer Features und besitzt daher ein verändertes User-Interface. Die Haupt-Features sind: Anzeige mit mehreren Fenstern; Auswertung von Indirektionen; Sie können ein mit niedriger Auflösung erstelltes Programm in hoher Auflösung debuggen (und umgekehrt); Quelldateien können im Debugger betrachtet werden; Disassemblierung in GenAm-Format mit automatischem Label-Generator zum Drucker oder auf Diskette; History-Puffer; Bedingte Breakpoints; liest auch Debugcode mit mehreren Hunks (wie von Lattice C); zeigt `_LVO`-Offsets für Library-Aufrufe an.

F.1.3 Integration

Die wohl größte Verbesserung ist die Integration aller Teile im gesamten Paket. Der Assembler und der Debugger sind vom Editor aus auf Tastendruck verfügbar. Der Assembler assembliert direkt in den Speicher; der Code kann dann im Editor ohne Diskettenzugriffe ablaufen. Falls erforderlich, kann die Debug-Information in Programme integriert werden, die in den Speicher assembliert wurden. Sie können dann vom Editor aus mittels Tastendruck assembliert werden.

Der Editor merkt sich Fehler und Warnungen, so daß sie mit `[A]`-`[J]` durchgegangen werden können. Wenn die Zeilennummer geändert wird, gehen die Fehler nicht mehr verloren (obwohl keine Neuberechnung angestellt wird). Nach einem Assemblerlauf, in dem ein Fehler auftrat, wird der Cursor in der ersten fehlerhaften Zeile plaziert.

Stichwortverzeichnis

A

Adresse 57
Adreßfehler 20, 102
Adreßregister 51
AmigaDOS-Fehlernummer 30
AmigaDOS-Header 73
Anfangsadresse 87
Assembler 11
Assembler-Module 16
Assemblieren 35
Assemblieren, bedingtes 63
Assemblierfehler 18
Asterix 47
Auflösung, niedrige 87
Ausdruck, numerischer 84

B

Backups 30
Befehl, illegaler 102
Bildschirm-Editor 11, 24
Bildschirmumschaltung 89
Binär-File 39
Bit-Test-Befehl 51
Blockpuffer 33
Breakpoints 83
Breakpoints, bedingte 90
–, einfache 90
–, permanente 90
BSS-Sektion 71
Busfehler 102

C

Chip Memory 71
Code, ausführbarer 70
–, linkbarer 70
–, positionsunabhängiger 55, 73
–, sofort ausführbarer 16
CODE-Sektion 71
Condition-Code 51

D

DATA-Sektion 71
Deallokation 97
Debug, erweiterter 56
Debug-Code 63
Debuggen 36
Debugger, symbolischer 77
Debugging-Information 54
Deinitialisierung 22
Disassembler 77
Disk-File 33
Druckertreiber 14
Druckertyp 33

E

Editor-Puffer 30
Entwicklungstools 12
Exception 77
Exec-Library 21
Expansion 21

Exporte 72
Extension 43

F

Fast Memory 71

G

Gadget 37
Grafik-Library 87

H

Hilfsbildschirm 34
History-Buffer 91
Hunk 84

I

Importe 72
Include-Datei 11
Indirektion 85
Initialisierung 35
Intuition 24
Intuition-Library 87

J

Justierung 57

K

Kickstart-Diskette 12
Kommentar-Feld 45

L

Label, lokales 49
Label-Feld 44
Langwortgröße 85
Langwortzeiger 102
Linker 1
-, Modus 54
Listing 54
Low-Lewel-Debugger 77

M

Makro-Expansion 54
Makro-Puffer 65
Makroentwicklung 65
Mnemonik-Feld 45

Module 16, 72
Modus, case-sensitiver 65
Move-Anweisung 21

N

Narrow 54

O

Objektmodule, linkbares 43
Offsets, relative 96
Opcode 94
Optimierung 55
Output-Handle 22

P

Prädekrement 86
Privilegverletzung 102
Programmzähler 21
Pseudo-Mnemonik 52

Q

Quellcode-Datei 18

R

RAM-Bereich, oberer 96
Register-Equates 49
Registerkorrektur 102
Registernamen 45
Relozierinformation 43, 73
Repeat-loop-Konstruktion 58
Requester 25, 30
ROM-Breakpoints 91

S

Sektion 71
Single-Steppen 21, 82
Source-Operand 51
Speicherlimit 24
Stack-Zeiger 45
Stand-alone-Assembler 41
Stop-Breakpoint 90
Stringabgrenzung 47
Supervisor-Modus 85, 101
Supervisor-Stack-Pointer 85
Symbol, reserviertes 65
Symbol-Optionen 96

Symboltabelle 55
Syntaxcheck 40

T

Tabulator-Taste 28
Text-Gadget 26
Trace-Bits 77
Trace-Exception 92
Turn-around-Zeiten 42
Typen-Überprüfung 55

U

Unterstrich 56
Usermodus 85

V

Voll-Screen-Editor 23
Voreinstellung 35

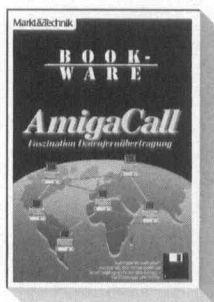
W

Warnung 56
Wortgrenze 50

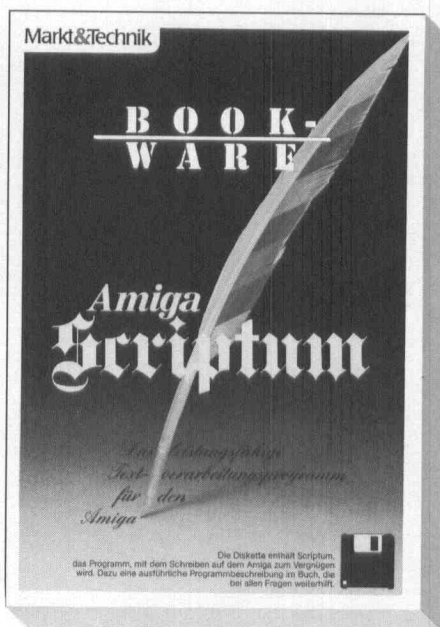
Z

Zähler-Breakpoints 90
Zeilen-Editor 24
Ziel-Operand 51
Zoom-Kommando 83
Zwei-Paß-Assembler 42

Bücher • zum Amiga

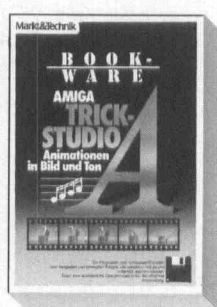


Atlantis
AmigaCall
Treten Sie ein in die faszinierende Welt der Datenfernübertragung. Kommunizieren Sie über Mailboxen mit erfahrenen Computer-Anwendern, die Ihnen bei Ihren Problemen weiterhelfen können, oder Sie erhalten auf diesem Wege leistungsfähige Public-Domain-Software. AmigaCall nimmt Ihnen die meiste Arbeit ab. Schließen Sie Ihr Modem oder Ihren Akustikkoppler an, starten Sie AmigaCall - und auf geht's.
1988, 133 Seiten, inkl. 3 1/2"-Programmdiskette
Bestell-Nr. 90716
ISBN 3-89090-716-4
DM 99,-*
(sFr 91,-*/öS 842,-*)



R. Arbinger/I. Krüger
Scriptum
Scriptum - das schnelle, leistungsfähige Textverarbeitungssystem für den Amiga. Ausführliche Bedienungsanleitung im Buch. Für alle, die auf dem Amiga Texte verarbeiten wollen.

1989, ca. 200 Seiten, inkl. 3 1/2"-Programmdiskette
Bestell-Nr. 90650
ISBN 3-89090-650-8
DM 79,-*
(sFr 72,70*/öS 672,-*)
* Unverbindliche Preisempfehlung



Atlantis
Trickstudio A
Ob Sie Computerfilm-Pionier sind oder Trickprofi, ob Sie von Walt Disney inspiriert sind oder einfach nur einen guten Lehrfilm für technische Abläufe benötigen: Mit Trickstudio A können Sie Ihre eigenen Trickfilme erstellen und diese mit Sound und Geräuschen untermalen.
Wie wäre es also mit einem Stummfilm-Slapstick, einem Krimi oder einem Werbefilm für Ihr Schaufenster? Dazu Ihre Lieblingsmusik oder digitalisierte Stimmen? Entwerfen Sie die Einzelbilder, z.B. mit Deluxe Paint, erstellen Sie eine Sounddatei und dann: Klappe - Film; die erste.
1988, 86 Seiten, inkl. 3 1/2"-Programmdiskette
Bestell-Nr. 90715
ISBN 3-89090-715-6
DM 99,-*
(sFr 91,-*/öS 842,-*)



HiSoft Devpac Assembler

deutsch

für die Amiga 500, 1000 und 2000

Mit Devpac 2.0 für den Amiga liegt Ihnen ein Entwicklungspaket vor, mit dem das Programmieren Spaß macht: ein integrierter 68000er Makro-Assembler, ein vollständiger Bildschirmeditor und ein mächtiger Disassembler/Debugger stellen ein maßgeschneidertes Paket dar, mit dem Sie jede Programmierhürde meistern.

Der Assembler/Editor:

GenAm ist ein schneller Makro-Assembler, der durchschnittlich 35000 Zeilen in einer Minute assembliert. Danach steht ein bereits ausführbares Programm bereit. Liegt ein Programmfehler vor, gelangen Sie unmittelbar in den Editor und können so komfortabel die fehlerhaften Eingaben korrigieren. Der Editor arbeitet mit Intuition – Sie müssen also nicht auf

Fenstertechnik, Mausbedienung und Menüs verzichten. Die Syntax des Assemblers ist Motorola-Standard. Es können sowohl linkbare als auch sofort ausführbare Dateien generiert werden. Beim Makro-Aufruf sind bis zu 36 Parameter erlaubt; Makro-Aufrufe und Includes können so tief verschachtelt werden, wie es der verfügbare Speicherplatz gestattet. Symbole besitzen eine Signifikanz von 127 Zeichen, lokale Labels werden unterstützt.

Der Disassembler/Debugger:

MonAm ist ein symbolischer Disassembler/Debugger, mit dem erstellte Programme inklusive aller Labels im Speicher untersucht werden. Das zeit- und nervenraubende Analysieren von Hexzahlen entfällt somit. MonAm bietet

auch die Möglichkeit, einzelne Befehle schrittweise nacheinander auszuführen und Breakpoints zu setzen. Durch Programmfehler entstehende Exceptions werden vom Debugger abgefangen, so daß es selten zu den berüchtigten »Guru«-Meldungen kommt.

Lieferumfang:

- 3 1/2"-Diskette
- deutschsprachiges Handbuch

Hardware-Anforderungen:

- Amiga 500, 1000 oder 2000 mit mindestens 512 Kbyte RAM
- Monochrom-Monitor
- mindestens ein Diskettenlaufwerk

Software-Anforderungen:

- Kickstart Version 1.1 oder höher



Hans-Pinsel-Straße 2
D-8013 Haar bei München



Unverbindliche
Preiseempfehlung

DM 149,-
sFr 135,-
öS 1490,-